The usefulness of case in plastic user interfaces

Robert Eurig Mitchelmore

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

July 2016

Abstract

This thesis addresses a problem that faces developers of applications for mobile devices. There is an ever-increasing number of mobile platforms and form factors in the world, and mobile developers have to build applications that can be used on as many of these as possible while still retaining usability. Furthermore, because of constraints put on the development process by the companies that develop the mobile platforms, there is an absolute requirement that the applications produced by the tool conform to the user interface guidelines for each platform.

To address this problem, this thesis uses the concept of "case", which is a phenomenon from natural language. In natural languages, case has many functions and plays a part in many systems. This thesis engages with case in one of these functions: it permits flexible word ordering. Case is used here to allow flexible ordering of elements within the user's dialogue with the machine. Case may either be useful because of some analogical process in the developer's head or because of deeper ideas in linguistic theory.

To evaluate this idea, a suitable case system was embedded in a tool and this tool was used in three distinct contexts. First, applications were built for three external companies. Second, a workshop study was done with external developers. Third, more external developers were given the tool for a longer period to produce an application of their choosing. These three contexts gave an excellent view into the use of the case system during the development of applications.

This evaluation showed that the kinds of functions that case describes are relevant to describing user interfaces; that it is possible to implement a plausible case system usefully in a software tool, at least for mobile development; that the case system when embedded within the tool can be used to build useful applications; and that case can be used and understood by developers other than the author.

Acknowledgements

The author was supported by the Horizon Centre for Doctoral Training at the University of Nottingham (RCUK Grant No. EP/G037574/1) and by the RCUK's Horizon Digital Economy Research Institute (RCUK Grant No. EP/G065802/1)

This thesis would not exist without the assistance, efforts and kindness of many people. These acknowledgements are not exhaustive: and if anyone feels they ought to be here but aren't, I beg that they will consider it as due to disorganisation, rather than to ingratitude.

I would like to thank my supervisors, Dr. David Golightly and Dr. Henrik Nilsson, who have with great good humour put up with far more chaos from me than anyone should have had to, and who gently guided me from producing baffling and incomprehensible waffle into producing something presentable. I would also like to mark my respect and gratitude to the memory of Prof. John Wilson, who also supervised this Ph.D for its first year until, very sadly, he became too unwell to do so.

I'd also like to thank the people involved with making the Horizon DTC happen, and the impressive dedication to interdisciplinary research that led to this project being accepted. When I initially presented this idea, shorn of evidence and exegesis, as something I wanted to pursue, it must have sounded outlandish indeed. I am grateful for the confidence and support that still thought it was worth pursuing. Prof. Chris Greenhalgh especially helped with the seed of the methodology and Prof. Sarah Sharples provided valuable one-on-one time at a point where the whole thesis-edifice was in danger of slipping into a morass.

My entire family have been very supportive both materially and morally. I am very grateful to them for this, and for putting up with the alternating phases of neglect and neediness that the process of writing instilled in me. I am also blessed with a wide and generous circle of friends, all of whom have been unfailingly supportive. I would like to make special acknowledgement (in no particular order) of Anna Clarke, who read chapters and kept me sane; Johannes Punkt, likewise; Pao, who among other things helped me read an important document in Swedish; Mark Porter, who shared LaTeX-related tribulations and who went far beyond the call of duty in hunting down obscure sources for me from the depths of the Bodleian;

and Claire, who shouted—she knows why this is important. I would also like to thank (again, in no particular order) colleague-friends from various parts of industry who have supported this work in various important ways: Robin Ellis, Matthew Munson, Malcolm Rooker, Cosimo Turroturro, Colin Williamson, Frantz Meckler, Mike Kelly. My study participants also came from this world and I am grateful to them all. Finally I would like to acknowledge the invaluable support from my colleagues—who are now my friends—in the 2009 Horizon DTC cohort.

This work has brought me into contact with many languages and given me some knowledge of the social situations around them. Therefore, I would also like to use this space, perhaps perversely, to mark my respect for everyone who has woken up one morning to find that they are the last speaker of their language, and with the knowledge that when they die an entire world of lexical, grammatical and poetic complexity will go with them. It is an experience that I do not envy them.

"The notion of a 'major language' is obviously primarily a social characterisation ...When linguists learned in 1970 that the last speaker of Kamassian, a Uralic language originally spoken in Siberia, had kept her language alive for decades in her prayers—God being the only other speaker of her language—they may well have wondered whether, for this person, the *world's major language was Kamassian."*

—Bernard Comrie, from the *Preface* to *The world's major languages*

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

In their 2012 "State of the Network" report, Cisco made the claim that by the end of that year there would be more mobile devices in the world than human beings. They were vague—perhaps artfully so—about precisely what constituted a mobile device in this report, but the basic premise of the statement is backed up by other large-scale market research (Lipsman and Aquino, 2013) which shows that people are increasingly using multiple mobile devices per person, and that these mobile devices are not identical either in form factor or in terms of platform. It is in developers' interests to make their applications available, therefore, on as many platforms and form factors as possible.

In the research community, this problem is part of the wider problem of "user interface plasticity". A plastic user interface is one that remains usable under changing circumstances, those circumstances consisting potentially of what kind of user is using the application, through what kind of device the application is being made available to the user and in what context the application is being used (Thevenin et al., 2003). The research community have created a number of tools for creating plastic user interfaces. These tools concentrate largely on flexibility, aiming to address very diverse kinds of devices and modes of interaction. One important concern of the research tools is adapting the "dialogue" between the user and the machine, the task structure and the orders in which the user's choices can be made, and a number of models have been built to assist in plasticising this dialogue (Paternò et al., 1997; Book and Gruhn, 2004). Thevenin et al. (2003) refer to the dialogue as the "keystone" of the user interface, and its plasticisation as of high importance.

Independently of the research community, the professional community around mobile application development have created a very large number of tools that aim at producing applications for multiple platforms and form factors. In 2012, Vision Mobile surveyed many of these tools and analysed them (Jones et al., 2012). The

concerns of these tools, taken as a body, differ from those of the tools produced by the research community. They address fewer platforms and form factors (for example, while several of the research tools target voice-activated devices, none of the industrial tools do). Instead, they put high value on developer productivity and on using and repurposing tools that developers already use, and on integrating well with third-party software.

Both of these approaches have shortcomings. The research tools operate under very unfamiliar sets of assumptions for mobile developers and do not take into account some very important factors for these developers, such as the structures of user interface guidelines. The industrial tools do not let the structure of the user's dialogue with the machine change easily as it moves between platforms and form factors. This thesis proposes to attack this gap by using the notion of "case", found in natural languages.

In comparison with the fairly recent importance of plastic user interfaces, the study of case is a very long-running discipline. Research into the nature of case was out of fashion for much of the twentieth century, because it was often seen as not being a phenomenon in its own right but merely as an expression of "deeper" phenomena. However, in the last 25 years it has become obvious that these explanations of case are insufficient, and that case exhibits strong patterns across different languages that argue for its right to be treated as a distinct phenomenon (Blake, 2001) that is not necessarily attached purely to language *sensu stricto* (Luraghi, 2009).

Case is interesting in the context of plastic user interfaces because one of the features it brings to languages that possess it is flexible word order in sentences. This flexibility of word order is, for example, one of the things that gives Latin poetry its distinct flavour (Clackson, 2008). The fundamental idea of this thesis is that the categories that case provides cross-linguistically to make word order flexible in sentences might also be usable to make the orders of users' choices flexible in plastic user interfaces for mobile devices.

This thesis, then, is of interest to the HCI engineering community, discussing as it does a novel mechanism for use in the engineering of plastic mobile user interfaces. HCI engineering approaches were chosen for this, as opposed to broader software engineering, because the existing work in the area of plastic user interfaces is in this discipline. HCI engineering refers to "the application of scientific knowledge and rigorous design methodology to reliably predict and, thus, help improve the consistency, usability, economy and safety of solutions to practical problems" (ACM, 2013, quoted by Blandford, 2013). While a linguistic approach would seem more appropriate to the analysis of the usefulness of case, there is no straightforwardly acceptable linguistic methodology that would allow the application of case to user interfaces to be evaluated.

1.2 Research questions

The thesis explores the application of case to plastic user interfaces by answering a series of research questions.

Are the categories delineated by case relevant to user interfaces? If the kinds of meaning that case delineates are not useful for talking about user interfaces, then any attempt to apply them will fail. If they are, then it should be possible to talk about user interfaces in terms of these meanings, and to use these meanings to help plan user interfaces.

Can a tool be built that embodies case for building plastic user interfaces? To be most useful in a development situation, the categories that case provides should be embodied in a tool that developers can use to build applications.

Can case as embodied in this tool be used to build useful applications? If the use of case cannot scale to use in the kind of applications that mobile developers build professionally, then its use is severely limited.

Can case as embodied in this tool be used by other developers? The author's experiences in the case studies can only be generalised if they are backed up by information involving the use of case by other developers. If only the author is able to use the case system, then it is not an effective tool.

1.3 Summary of method, results and conclusions

1.3.1 Method of evaluation

To evaluate the usefulness of case to commercial development, this thesis outlines one set of case studies and two co-operative evaluation studies (Wright and Monk, 1991) that were undertaken in three contexts. Ethical approval was sought and granted for all work involving participants.

1.3.1.1 Case studies

Three companies were approached to submit commissions for cross-platform mobile applications. The applications were not prototypes but finished applications intended for release. Meetings throughout the process were recorded. Clients were asked to comment on the quality of user interface elements that were implemented using the case system and to comment on the use of case in the development process.

1.3.1.2 Developer Workshop

Seven developers were recruited through the first author's professional network for a development workshop. Developers worked in pairs. The workshop was recorded in both video and audio, and the applications created were collected. The workshop opened with a training session followed by a question and answer session. After this, participants were set to building a small application for a hypothetical client. At the end, participants undertook a focus-group review of the tool.

1.3.1.3 Long-term development study

Seven further developers were recruited through the first author's professional network for a longer-term development study. All were commercial software developers who had commercial mobile development as part of their professional practice. Participants worked remotely, receiving the same training material as workshop participants, but presented in writing on paper rather than verbally. Participants were asked to spend a minimum of two days working on this problem. Developers were asked to comment on the quality of the user interfaces that they built as they went, and answer further questions at the end. Where the developer and the first author were co-located, audio recordings of discussion sessions were made. Otherwise, email or chat transcripts were saved. Applications were submitted back and checked against the platform style guides by the first author.

1.3.2 Results

Three key points emerged from the evaluation exercises. First, the parts of the interface that the case system produced were of a quality comparable to the rest of the interface, although the quality of the user interface as a whole was not as good as it would be had it been made with platform-specific tools. This was especially underlined by the participants who counted web design as part of their professional practice. Likewise, there was much of the interface that case did not touch, and the quality of resulting interfaces as a whole still owed a great deal to the manual intervention of developers. Second, the plasticity effected by case noticeably reduced the work that needed to be done to produce a user interface that conformed to the style guides for the platforms being targeted. This reduction in work was most evident at the early stages of application design, when the structure of the application was being laid out. Third, case was not found to be universally applicable across the whole domain of mobile development. Instead, it was most useful in data-driven applications, where users were interacting with a dataset, rather than games or communications applications.

1.3.3 Conclusions

In commercial contexts, case was useful in effecting plasticity in user interfaces for mobile applications as they were built. The case system was used readily by developers in both the workshop and the longer-term studies, who felt case would be of use in their professional practice.

1.4 Structure of thesis

Chapter 2 summarises the relevant work on plastic user interfaces for the remainder of the thesis. It summarises the problem, looks at the vocabulary and models that inform research tools for building applications plastic user interfaces and introduces both research tools and tools from industry for building applications that are at home on multiple platforms.

Chapter 3 describes what case is and what it does in natural language, illustrating it with examples from Latin literature. It then uses a sample of twenty languages from all over the world as an illustrative survey to demonstrate that case is used in similar ways in different and unrelated languages. It then summarises work on the patterns that are found in case systems and mainstream theories of case that admit meanings for cases (rather than those theories which simply consider them as a syntactic convenience). Then it provides an example of comparing two user interfaces with two different ordering conventions in terms of case.

Chapter 4 describes the methodology for the design of the tool that embodies a case system and for the evaluation of the case system through the medium of this tool.

Chapter 5 describes the implementation of the tool, beginning with the rationale for the choice of cases to use in the case system. It then looks at how those cases should be realised in the concrete user interface, how a dialogue component and stylesheet mechanism can be defined in terms of case, and finally describes the software architecture and how the dialogue component of the application being built interacts with the rest of the software components needed for that application.

Chapter 6 describes three case studies in which the tool was used to build applications for three very different small companies. It then evaluates the contributions to the development process made by case.

Chapter 7 describes two studies that evaluated the usefulness of case to other developers. The first study was a workshop study in which developers eval-

uated the usefulness of case in their professional practice in general. The second was a more focused study in which developers each developed an application and then commented specifically on the usefulness and limitations of case in their experiences.

Chapter 8 synthesises answers to the research questions above from the process of the development of the tool, the results of the case studies and the results of the two studies with other developers.

Chapter 9 pulls together the contributions of the thesis and suggests both some implications of the thesis and future directions in which the research could be taken.

1.5 Typographical and lexical conventions

Several typographic and lexical conventions are used throughout this thesis.

A paragraph introduced by small capitals such as this one is a definition of a term or concept.

A paragraph introduced by italics such as this one is a description or a discussion, or answers a question.

In chapter 3, several analyses of sentences are presented. In keeping with standard practice these are presented in a three-line format, for example:

(1) Caecilius est in horto
Caecilius.NOM is in garden.ABL
Caecilius is in the garden

The first line is the text in the source language, split into words. Beneath each word is a literal translation of that word into English, along with any relevant extra information that would otherwise be lost in translation. In this thesis, the only such information presented is the cases that individual words are in. Beneath both of these are an idiomatic translation of the phrase into English.

Cases have long and Latinate names which almost universally end in the suffix *-ative*. For conciseness, both in the analyses of sentences and in tables laying out the case forms of various words, the names of cases are abbreviated to just their first three letters presented in small capitals. For example, "nominative" is abbreviated as Nom and "accusative" is abbreviated as Acc.

When languages are being discussed, some words and phrases begin with an asterisk (*). These are words or phrases that are not attested: they were never

spoken or written. These phrases should be considered as provisional, for the sake of discussion; but they should not be considered as part of a real language. This convention is used either for reconstructed words or phrases in an extinct language that may never have been valid, or for phrases that are ungrammatical in living languages.

Finally, there is some terminological collision between the disciplines of linguistics and computer science. In this thesis, this affects the use of the words "syntax", "semantics" and "grammar". Except where the computer science meaning is explicitly specified, these words should be assumed to bear their linguistic meanings. This especially means that other frameworks that have a "grammatical" basis, where "grammar" is meant in the computer science sense, are not necessarily comparable to the work done in this thesis. A good example of this terminological collision is Payne and Green's (1986) "Task-Action Grammars".

Chapter 2

Frameworks, tools and standards

2.1 Introduction

The research community has coined the term 'plastic user interfaces' to refer to interfaces that can move between contexts of use while maintaining usability (for some reasonable definition of usability). This chapter argues that this concept is important for developers of applications that run on mobile devices and summarises the solutions that are currently available.

The chapter begins with an overview of the nature of a major problem that developers for mobile devices face: that to capture a large user base it is necessary to work on multiple platforms (for example, Android and iOS) and multiple form factors (such as tablets and smartphones), and that developers' conformance to the user interface idioms for those platforms is enforced by the companies that create those platforms. Therefore, developers are forced either into creating multiple user interfaces, or a single plastic user interface that adapts to the idioms of the platform and form factor on which it finds itself.

Second, the chapter looks into conceptual models and frameworks that have been used to talk about plastic interfaces by different people. These models do not form a single coherent whole, they are not parts of a single large overarching model: rather, they are ways of examining and comparing different tools in different ways, or are guides to building tools, that can be used in different situations as and when they are helpful.

Third, the chapter examines the efforts that the research community have put into building tools that can be used to create new applications with plastic user interfaces. It examines five tools, summarises how they work and their underlying philosophy, and compares them. Among other parts of the interface, these tools allow the task structure of the application to change to suit circumstances.

Fourth, the chapter gives an overview of an illustrative sample industrial approaches and tools to address this problem. There are an enormous number of these tools, albeit generally more pragmatic and restricted than those from the

research community. The chapter classifies and compares this sample of tools. It also surveys a number of tools from industry that were not designed with mobile interaction in mind, but that have been repurposed for application in mobile technologies. It also considers tools from the web community: the availability of web applications on mobile devices is increasingly important, and the web community have created a series of standards and design approaches that make it possible. The line between a web application and a mobile application is increasingly blurry: many of the industrial tools for cross-platform development use web technologies. This means that techniques usable by web designers are increasingly useful to application designers also. These, too are compared with one another and with the research tools. Specifically, it is highlighted that unlike the research tools these industrial tools do not provide adaptation of task structure.

Fifth, it points out that although the industrial tools provide no dialogue adaptation, a number of them do use declarative notations to structure their dialogue. It presents notations in use for dialogue structure in this field, both from the research community and the industrial community are presented.

Finally, it argues that none of the tools from either the research or industrial spheres are adequate to address the problem that application developers face, and suggests that case may help to bridge the gap between research and industrial tools to a useful degree.

2.2 The Mobile Platform Problem

At the beginning of 2013, comScore, a market research company who specialise in analysing people's uses of digital technologies, released a report about the kinds of smartphones and tablets in use over the previous year (Lipsman and Aquino, 2013). There is no single dominant software platform among these devices. In their worldwide analysis, they identified six major platforms.

Android is an operating system developed and distributed by Google, and used by a number of equipment manufacturers. As of the publication of the comScore report, it was the most used operating system for smartphones in both the US and the five analysed countries in the EU. It was also the most used operating system for tablets in the US. Figures for tablet operating systems in the EU were not given.

iOS is Apple's proprietary mobile operating system, in use on the company's smartphone and tablet products. It was the second most used operating system for smartphones in both the US and the five analysed countries in the EU, and second most used tablet operating system in the US.

Blackberry OS is a family of related operating systems developed by Research In Motion for use on their Blackberry phones and tablets.

Some Windows operating systems are available for phone and tablet use from Microsoft.

webOS is an operating system currently developed by LG for their smart televisions. Previously, it was owned and developed by Hewlett-Packard, who built smartphones and tablets that used it.

Symbian is an operating system that used to be used on Nokia's high-end phones before they switched to using Windows in 2011. There were no tablets running Symbian. Smartphones that run Symbian are still a major part of the smartphone ecosystem in the EU.

The challenge to developers is not only in the number of platforms that one needs to develop for in order to cover all smartphone users, but the volatility of that set of platforms. Blackberry OS, Symbian, and Windows, all of which are currently minority players, have all been the dominant platform at one time or another. In addition, new platforms keep emerging, such as the Firefox OS backed by the Mozilla Foundation and Tizen, backed by Samsung, Intel and others. Others disappear suddenly: in 2009, Nokia's great hope was the Maemo operating system, which they later unceremoniously abandoned to be developed by the community and changed their focus to working using Windows Phone.

To add further to the difficulties of the developer of mobile applications, many people use multiple platforms and form factors and expect the same services to be available on all of them. comScore, in another report (Lipsman, 2013), refer to this as the "multi-platform majority". They report that in the US, at least, over half of all users of mobile technologies now do so in a multi-platform way. Even in the case of a user who uses only a tablet and a smartphone, it is not uncommon for these devices to be running different operating systems (Lipsman and Aquino, 2013, p. 30).

These circumstances create a situation where cross-platform and cross-form-factor development is a problem that needs to be taken seriously by anyone who wants to create an application for a mobile platform.

2.3 Being at home on a number of platforms

To be at home on a mobile platform, the application needs to conform to the user interface guidelines published by the developer of the platform. Whether these guidelines in fact enforce or promote usability is in some ways moot.

Software distribution for mobile platforms is usually very centralised. For each major platform there is an online application store run by the developer of the

operating system, and in nearly all cases application delivery is done from there. In the cases of iOS and of the versions of Windows that run on mobile devices, all application delivery (with the exception of some enterprise software) is done from the platform's store. In the cases of Android and Blackberry, things are a little more complicated, but most end-user downloads are still done through the operating system manufacturer's store.

Because of this, Google, Microsoft, Apple and Blackberry effectively have a veto on which applications run on their mobile platforms; and all of these companies emphasise the importance of their own user interface guidelines. At time of writing, Google does not enforce their guidelines, and there is no manual checking of Android applications at all (Google, Inc., 2013c); however Microsoft and Research in Motion both at least check the navigation structure of all applications (Microsoft Corp., 2013a; Research in Motion, Inc., 2013), and Apple perform careful checking of many aspects of the interface against their UI guidelines (Apple, Inc., 2014).

This means that adhering to each platform's user interface guidelines is effectively forced onto developers for mobile devices. Therefore, making sure that an application complies to these guidelines is an important part of the engineering of the user interface. This is the point at which the mobile developer's direct interests intersect with the research area of "plastic" user interfaces.

2.4 Some definitions and terminology

The terms "cross-platform" and "cross-form-factor" are not precise. To clarify both the problem space and the aims of the various research tools, Thevenin et al. (2003) gave definitions for a number of terms. Their definitions are paraphrased here.

A platform encompasses the resources available to the application. This includes both software resources such as the facilities that the operating system provides and hardware resources, such as screens and input devices and their characteristics.

A target is a set of three specification elements: the class of user which will use the system, the platform upon which they will use it, and the physical environment in which they will use it.

A multi-target interface is an interface which is designed to be usable for more than one target. This does not mean that adaptation has to occur to differences in all three elements in the target; an interface that adapts to different platforms but not to different users or to different environments, or an in-

terface that adapts only to different users but remains bound to one platform and environment is still a multi-target interface.

A MULTI-PLATFORM INTERFACE is a multi-target interface which adapts to different platforms, but only to one class of user and one physical environment.

A PLASTIC INTERFACE is a multi-target interface which maintains usability across its different targets.

TO PLASTICISE AN INTERFACE is to take a non-plastic interface and make it plastic, or to derive a plastic interface from it. Plastic user interfaces are, of course, not at all confined to mobile platforms. This thesis, however, concentrates on mobile platforms because of the nature of the problem given above.

2.5 Models and frameworks

There are a number of useful models and conceptual frameworks for talking about the tools that are used to build plastic interfaces, and the software infrastructure that sits underneath them.

THE ARCH MODEL (Bass et al., 1992) or THE SLINKY is a model for describing software architectures for interactive systems. It has the specific aim of "guid[ing] developers in designing a run time system that makes possible the efficient and robust management of change during the life cycle of a user interface." (Sheppard, 1992, p. 31) It was used by Thevenin et al. (2003) to discuss the levels at which plastic interfaces are able to adapt the interface, and it is in this way that it is used in the analysis below.

The Arch model is illustrated in figure 2.1. It lays out five areas with which software architectures for interactive systems must concern themselves. The first area is the "functional core": the components in this area concern themselves with the application's logic, and provide the basic functionality of the application. The second is the "functional core adaptor": the components in this area are responsible for selecting which functionality to expose to the user as the infrastructure that supports the interactive system changes. The third is the "dialogue controller": components in this area are concerned with task structure and the order in which the user should be able to do things. The fourth is "logical presentation": components in this area are concerned with deciding which kinds of "interactors", or user interface elements, to use. The fifth is "physical presentation": components in this area are concerned with the actual appearance of the interactors.

The Arch model as used in this thesis has been extended by the addition of a "platform" area. This area corresponds to what Thevenin et al. call

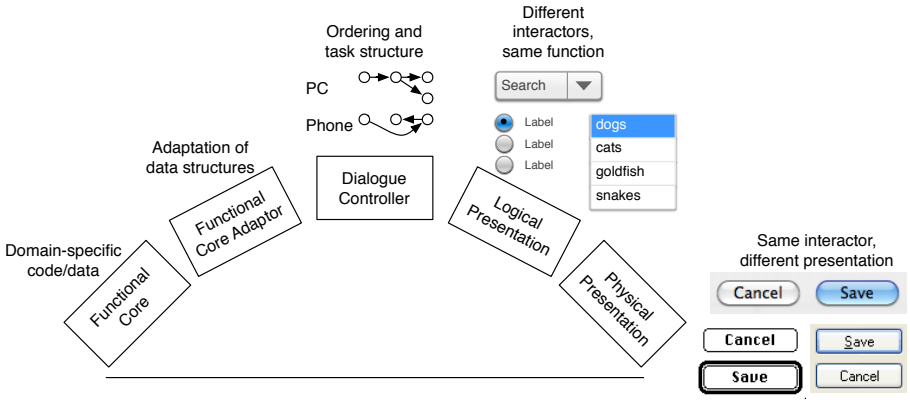


Figure 2.1: The Arch model (after Thevenin et al., 2003)

"technical adaptation", sitting beneath the physical adaptation layer. It is responsible for the infrastructure necessary to put the physical components on the screen, and to provide them access to the input devices. It will be used extensively in the work that follows as a taxonomy of the layers on which adaptation takes place.

Model-View-Controller ("MVC") is a collective name for a large and widely-used family of software architectures for interactive systems. These architectures share the idea that an interactive system is built out of three fundamental kinds of component (models, views and controllers) but differ considerably in their elaboration of this idea.

The first of these three kinds of software component is the "model". Models are responsible for providing data access. The second is the "view": views are responsible for taking the data from their corresponding model and displaying it to the user. The third is the "controller": controllers are responsible for reacting to user input and updating the data in their corresponding model appropriately. A pure MVC system is built of "triads", collections of three components where each triad consists of one model, one view and one controller.

The first, and simplest, elaboration of the MVC idea emerged from Xerox PARC as part of the Smalltalk system (Reenskaug, 1979; Krasner and Pope, 1988). In the Smalltalk implementation, one controller at a time could be active, and input from the system's "sensors" (Krasner and Pope's term for various input devices including, but not limited to, mouse and keyboard) went directly to the controller. The controller received mouse position changes, key presses, and mouse button presses. The controller then interpreted this input and updated the data in the model accordingly.

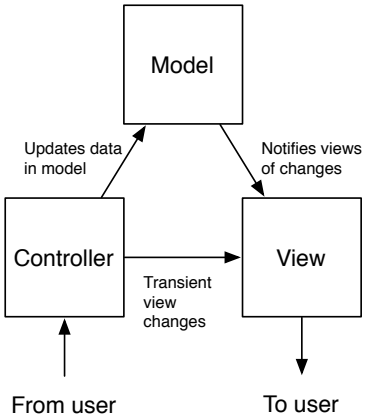


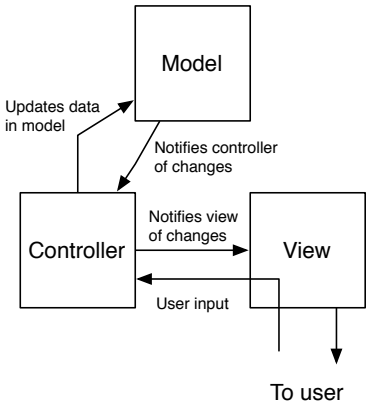
Figure 2.2: MVC in Smalltalk-80 (simplified from Krasner and Pope, 1988)

In this variant of MVC there is a strong but implicit binding between the view and the controller: specifically, the view and the controller must have consistent internal representations of what is being displayed on the screen. If, for example, the view displays some text at a given position on the screen with the expectation of the user being able to edit it, then the controller must also know where this text is so that it can interpret the input from the mouse.

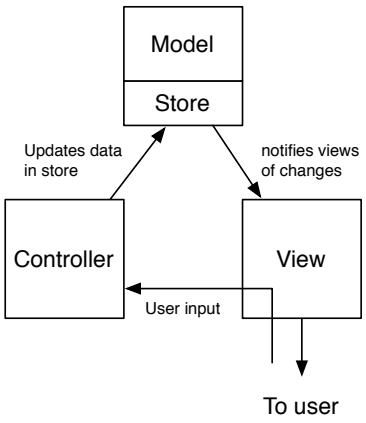
More modern variants of MVC send user input to the view first, so that it can interpret the sensor input according to what it is displaying, thus making this coupling looser. Two modern variants which exhibit this behaviour (those used in Apple's and Sencha's development tools) are illustrated in figure 2.3. In Apple's variant, the view provides the controller with user interface events in terms of the interactors in the view, for example "the user pressed the 'Print' button". In idiomatic Sencha Touch, this is taken even further: the view interprets the user's intent, and the controller contains actions which enact that intent, for example "the user chose to print".

MVC is relevant here because it is a widespread architecture family that is extensively used in industry, as shown by the analysis of software tools below.

Model-driven engineering ("MDE") is an approach to software engineering which emphasises the importance of formal domain-specific models (not to be confused with the "model" component of MVC) and the transformations between these models (Schmidt, 2006). By using domain-specific models, designers of systems can express meanings in terms of their domain knowledge: "some of the alleged benefits of [domain-specific languages] stem from



(a) MVC in Apple's Cocoa and Cocoa Touch (after Apple, Inc., 2013)



(b) MVC in Sencha's Ext.js and Touch (after Sencha, 2012)

Figure 2.3: Two modern variants on MVC

the fact that there is a close, intentionally *[sic]*, direct link between the program and the modeled reality in the domain." (Estublier et al., 2005, p. 71)

The first element emphasised within MDE is the domain-specific model (see Favre, 2004a, for an in-depth discussion of what a model actually is in MDE). A domain-specific model is expressed in a domain-specific language (a "DSL"). A DSL is a language—either textual or graphical—which uses familiar terminology and which enforces domain-specific constraints on the program. An example of such a language is LOGO. LOGO is a well-known DSL for control of a drawing robot called a "turtle", or for a simulation of such a robot(Papert, 1980). This language uses familiar terminology: the pen which the robot uses to draw on the paper is referred to as the "pen" in the language, and the robot is referred to by the same name inside the language

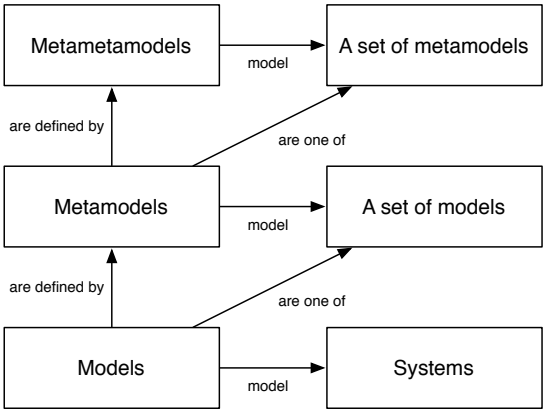


Figure 2.4: Models in MDE (after Estublier et al., 2005)

and outside. This language also enforces domain-specific constraints: it is not possible, for example, to try to get the robot to leave the ground or to draw in a colour for which it does not have a pen.

Not all DSLs are part of an MDE approach to building software. LOGO has no claims to MDE. Neither do many other major DSLs, such as Hypertext Markup Language ("HTML", a DSL for authoring hypertext documents, the current version of which is defined by Berjon et al., 2012), the Structured Query Language ("SQL"; a DSL for querying and updating databases, defined in ISO/IEC 9075:2011), or the Open Telecom Platform ("OTP"; a DSL for building resilient distributed telephony applications, described by Torstendahl, 1997). The difference between a non-MDE DSL and one for use in MDE is the existence of a "metamodel". A metamodel is a model which models a set of models, and thus "defines the language for expressing a model" (Object Management Group, 2002, p. Glossary-10). In a hypothetical model-driven version of LOGO, for example, where a program in the LOGO language is a model of a drawing done on paper, a metamodel might be expressed in terms of the capabilities of the modelled robot.

Since metamodels are models in their own right, in MDE they are built in a DSL for modelling models; this DSL is in its turn modelled in a "metametamodel" which defines a language for creating metamodels (Object Management Group, 2002, p. Glossary-10). The relationships between systems, models, metamodels and metametamodels are summarised in figure 2.4.

The second element emphasised within MDE is that of transformations between models. Here, as Favre (2004b) puts it, the MDE literature "suffers from a lack of agreement on terminology, especially when talking about transformations." (p. 3). Since, however, in MDE "everything is a model"

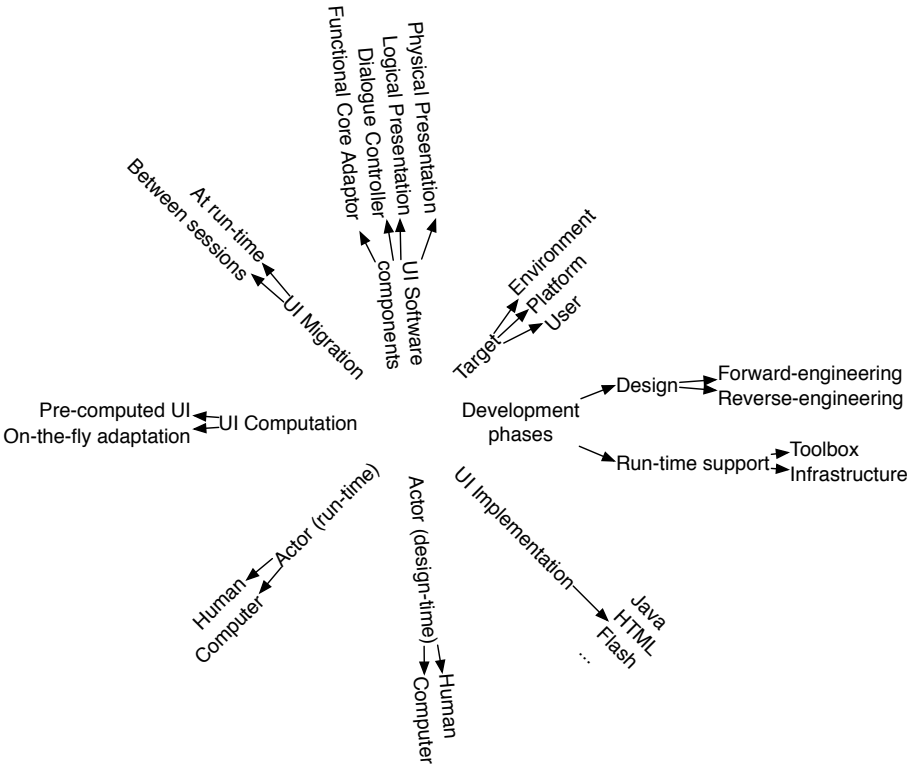


Figure 2.5: The Plastic UI snowflake (after Thevenin et al., 2003).

(Bézivin, 2004, p. 21), for the purposes of this description transformations can simply be considered as models which model such a transformation, and thus have their own metamodels and metametamodels.

The MDE approach is a major one for the area of plastic user interfaces; in the research tools given below, which aim to build entire applications with plastic user interfaces, it is dominant.

The Plastic UI snowflake is a taxonomic framework presented by Thevenin et al. (2003) that forms half of their reference framework for the development of plastic user interfaces. It is intended to be used "for characterising existing tools or for expressing requirements for future tools". (p. 32). Each branch of the snowflake characterises a number of related design points for existing tools, or a number of decisions which need to be made in the case of new tools. The snowflake is illustrated in figure 2.5.

The issues it addresses are:

What software components are capable of adaptation. These are expressed in terms of the Arch model.

Kinds of target. Thevenin et al. note that they are unaware of any tool which addresses all three kinds of target (environment, platform and user) simultaneously, but give examples of tools that adapt based on each of the possibilities.

Development phases. Some tools are only active during the design of a system, some only at the time the application is run, and some have components which are active in both phases. If the tool is active during the design of a system, it may be creating an entirely new system through forward engineering, or reverse-engineering an existing system to plasticise its user interface. If the tool is active at run-time it may provide either a reusable infrastructure or a toolkit of small, reusable parts.

How the interface is implemented. There is a very large number of tools for making the interface available to the user. The limitations of this implementation method may affect the nature of the tool being classified.

Which actors are in charge of adaptation. The interface can be adapted both during design and when the application is run. At either stage, either a human, a software tool, or a combination of the two can be responsible for the process of making the interface usable on a target.

When the computation of the final UI occurs. In some tools, the final UI is generated before it is ever seen by a user. In others, the final UI is generated on the fly, as it is needed.

Whether the final UI can migrate between devices, either in the middle of a user's interaction with the system or between those interactions.

The plastic user interface snowflake will be used in the work that follows as a way of comparing the scopes of different tools for building plastic user interfaces.

2.6 Research tools

The models above are useful for talking about and comparing the behaviour of tools, but they are not software tools themselves. They do not directly address the problem of how a mobile developer can build an application that is at home across multiple devices. In order to produce actual interfaces, they need to be embodied in software tools that can be used by developers.

The research community has created a number of tools around the idea of multi-target and plastic interfaces. The ones whose capabilities are summarised here are the ones that are forward-engineering tools (in the terminology of the Plastic UI snowflake) and are for the creation of new applications. There do exist

others, for reverse engineering of existing interfaces (such as Vanderdonckt et al., 2001), for creating compositions of existing content (such as Gabillon et al., 2011) or for building interfaces out of components that themselves manage plasticity (Calvary et al., 2005) but they are not within scope here because they are either not forward-engineering tools or not tools for the creation of entire applications where the concerns of plasticity are dealt with at an application level.

UIML is a model-based application of XML for building interfaces for multiple devices. It aims to make no assumptions about which toolkits will be used to make the interface available. It also aims to be easy to learn for people who are not developers (Phanouriou, 2000, ch. 3). Here, the application of UIML to mobile application development is described. It can also be used for the development of other kinds of user interfaces, such as voice-driven ones.

An interface designed in UIML consists of two basic parts. One of these parts describes the structure of the interface. The other defines how that interface will be realised on different platforms and form factors.

The section that defines the structure of the interface defines a logical outline of the interface and how different parts of that logical outline map to actions in the software that backs the interface. It also contains styling hints for how the interface should look on different platforms. The outline of the interface is a tree, where related interface elements are grouped.

When the interface is actually instantiated on a system, each element in the tree is matched up with a corresponding 'peer' physical component that provides the required features. This peer component is styled according to the hints that the developer gave. These physical components are then grouped in a way that reflects the tree structure of the interface. The system tries to show related interface controls together: the smaller the screen space, the smaller the subtree of the interface it can show at once. If it needs to, it adds navigation controls to let the user navigate through multiple pages of interface.

When the user interacts with the physical component that the system has chosen, the thing that the user did (such as 'click') is looked up in the stylesheet to turn it into a semantic event (such as 'activated an input field') and the appropriate action in the software that backs the interface is called.

Plasticity is effected in this tool in three main ways. The first way is the separation between the logical and the physical layers. The developer is not required to design the mappings between the physical and logical layers: a number already exist (such as Luyten and Coninx, 2005; Binnig and Schmidt,

2002; the Java renderer by Phanouriou, 2000; and open-source renderers for Java's Swing and Python's wxWidgets support). The second way is that the developer can provide different styling hints for different platforms, and so can manually adapt their application stylistically to different platforms. The third way is that the system adapts the dialogue between the user and the machine based on the screen size.

ARTStudio is a tool for building multi-platform applications in a model-driven way (Thevenin, 2002). It uses a process of "reification", which Thevenin et al. (2003) describe as being a process of transforming a model into another which is closer to executable software. An abstract model can be transformed into executable software by a series of reifications. ARTStudio allows building of applications for desktop computers and for Palm's Pilot personal digital assistants.

ARTStudio reifies in three phases; it starts with a domain model and a hierarchical task model. From these it generates abstract interfaces, from these it generates concrete interfaces, and from these in turn it generates the executable software which forms the final user interface. The relationship between the domain model and the task model is not fully explained either by Thevenin (2002) or Thevenin et al. (2003).

The task model is defined using Paternò et al.'s ConcurTaskTrees (1997). Abstract interfaces consists of a set of workspaces. On platforms which allow multiple overlapping windows to be visible at once, each workspace corresponds to a window; on platforms where only one full-screen view is visible at a time, such as most mobile devices, a workspace corresponds to such a view. There is one abstract interface per platform.

After the reification of the task model into an abstract interface, there is a one-to-one correspondence between workspaces and tasks. If a task has subtasks, then its workspace contains the workspaces of its subtasks, and is called a compound workspace. If a task has no subtasks, then its workspace contains no other workspaces, and is called an elementary workspace. The developer can edit the allocations of workspaces to tasks, and change how they are grouped; for example, a subtask can be removed from the workspace of its parent task and made into an independent workspace. Since each platform has its own abstract interface, different platforms can have their workspaces structured differently.

To generate a concrete interface from an abstract one, ARTStudio uses a model of the platform for which the concrete interface is being generated. This model contains information about the screen of the device and how

it manages its windows; a description of the programming language used to implement the interface; and information about each of the interactors which is available. The information about each interactor contains what data types it is able to edit, whether it acts as a mechanism for switching between workspaces, and the system and human resource cost of the interactor. These costs are estimated based on the on-screen area of the interactor.

To generate a concrete interface, ARTStudio performs a constraint satisfaction algorithm, the precise nature of which is not specified. Workspaces become either windows or "canvases" (which are not defined, but seem to refer to a container which opens inside a window). The concepts in the domain model which are involved in the workspace are mapped to interactors based on their data type and their importance using ARTStudio's model of the platform for which the concrete interface is intended. The developer is then free to edit the concrete user interface.

The plasticisation of the interface comes from the adaptation of the various components by the platform model; any part of the interface can be transformed by the platform model, or by the developer's explicit interference, to be more appropriate to the platform on which it finds itself.

Multimodal TERESA (described by Paternò et al., 2008) is a model and transformation driven tool for building multimodal applications targeted to multiple platforms. A multimodal application is one which can use multiple modalities.

Modalities are to do with senses: a modality is a sense that the human can use to communicate with the computer, or a sensor that the developer of the software can use to communicate with the user. For example, an application that uses a visual modality to present information to the user does so through graphics, or something which the user apprehends through vision. An application that uses an auditory modality to receive information from the user does so by processing sound input from the user. An application which uses an olfactory modality to present information to the user does so by generating meaningful smells (Bodnar et al., 2004; Brewster et al., 2006).

Multimodal TERESA concentrates on visual and auditory modalities. It allows building user interfaces which target any number of: interactive television (visual modality); VoiceXML (Oshry et al., 2007) telephony interaction (auditory); direct interaction using XHTML (the Extensible Hypertext Markup Language) and scalable vector graphics (visual); form-based XHTML on desktop computers (visual); minimalist XHTML for mobile devices; graphical and gestural interfaces for Microsoft's Windows Mobile

platform (visual and gestural); and graphical interfaces with vocal cues for desktop computers (visual and auditory).

Paternò et al. state that TERESA is designed to be used in a top-down fashion, beginning with abstract concepts and working towards executable software. This process begins with a hierarchical task model (expressed using the same ConcurTaskTrees notation as ARTStudio). From this, an "abstract UI" is generated by transformation. An abstract UI contains a number of "abstract presentations" along with an abstract view of their contents. A presentation is "a set of interaction techniques available to the user at a given time" (Berti et al., 2005, p. 42). An abstract presentation is one where the contents of that presentation are defined in terms of what they permit the user to do, rather than in terms of how they will be realised in the physical user interface. The abstract UI also contains an abstract view of the connections that the user will be able to use to move between these presentations.

Each element of an abstract presentation can either be a kind of action to be performed (such as ("permit the user to edit", "permit the user to select one of a set of alternatives", "output") and a kind of information for this action to happen to (such as "text" and "number"), or a group of these abstract actions annotated with a "composition operator" that identifies what kind of grouping it is. This operator represents a generic logical connection between the elements within the group; a one-to-many relationship where a change in one element within the group affects a number of others; that the elements within the group are ordered in a specific way; or that some elements within the group are more important than others.

TERESA maps each element of the task model to one or more abstract UI elements. It then uses the ordering relationships from the task model to decide what presentations to create and which presentations to put each element in. For example, if two elements of the task model are marked as being concurrent, then they can be displayed side by side, and thus can be put in the same presentation. If two elements are marked as sequential, they must be put in two different, but linked presentations.

From the abstract UI, a "concrete UI" is generated. The concrete UI is a model which contains all the necessary information to generate a "final UI". A final UI is something that can actually be executed on the target platform to display the user interface.

The nature of concrete UIs necessarily varies greatly across different platforms. Paternò et al. give details of the different kinds of concrete UI models and how they are transformed from abstract UI models and to final UIs. Again it is this process of transformation that provides the plasticity, adding

platform-specific features to the interface and making decisions based on the capabilities of the platform.

MARIAE (Paternò et al., 2009) is the successor to TERESA, but does not address entirely the same problem. MARIAE concentrates on engineering multimodal and multi-platform user interfaces for interacting with web services. A web service is a set of functions which run on a server and can be accessed from a client application. These functions are made accessible to the client through some kind of web technology, generally XML ("eXtensible Markup Language") over HTTP ("HyperText Transfer Protocol") in such a way that a client can submit a request and be returned a result. The methods that the client should use to access these functions is defined in a machine-readable format called WSDL ("Web Service Definition Language"; described by Booth and Liu, 2007).

The definitions of the web services can be used in two ways. MARIAE can generate a ConcurTaskTrees model from a WSDL specification, inferring the nature of the tasks and subtasks from the parameter and return types of the functionality provided by the web service. Alternatively, the developer can create a task tree for the application that they are trying to build, and then annotate parts of the task tree with their corresponding functionality in the web services. Using the latter approach, functionality from multiple web services can be integrated into a single task tree.

After the task tree exists and is annotated, the abstract, concrete and final user interfaces are generated through a transformation process similar to that of TERESA. The abstract and concrete user interfaces are expressed in an application of XML which maintains the annotations applied to the task tree throughout the transformations, so that the resulting final UI remains bound to the web services as initially specified.

DiaGen (Book et al., 2006) is a tool for adapting web applications to different sized mobile screens by dynamically generating dialog masks for web forms. A web form is here defined as a set of user interface components which accept user input of some kind and which can potentially all be displayed on screen simultaneously if the screen is large enough. A dialog mask for a given form and screen is a subset of the components on the form such that all the components in the mask can fit on the screen at once.

DiaGen adapts forms to smaller displays by creating a set of disjoint dialog masks so that no component is in more than one mask, then displaying the masks in sequence. It does this on the server which is providing the web application to the mobile device, rather than on the mobile device itself.

When a request for a form reaches the server, the DiaGen system first identifies a profile for the kind of device which made the request. This profile includes information such as screen resolution, screen size and what user interface elements are supported by the browser on the device. The DiaGen system identifies the kind of device by using the device's User Agent Profile (Wireless Application Forum, 2001) information if it exists. If this information is not provided, the DiaGen system makes an educated guess as to which profile to apply based on the name of the browser software in use on the device.

When the client device and its capabilities have been identified, the DiaGen system performs adaptation at the logical level of the Arch on the definition of the form, identifying which interactors should be used for which form element. The form is defined using a small extension to the XForms form modelling language (Boyer, 2007). Once the interactors to be used to render the form are known, DiaGen begins laying out the form elements into the dialog masks. The sizes of the dialog masks are known from the profile of the device identified earlier, and a rough idea of the sizes of the interactors can be worked out based on the nature of the interactor and the number of characters it allows the user to input. In general, DiaGen attempts to lay out the fields in the same order as they are specified in the form definition. When a dialog mask becomes full, the system creates a new, empty dialog mask and starts laying out interactors into that.

Some metadata in the form definition can affect the order in which interactors are laid out: for example, if the XForms grouping constructs are used, then the interactors inside the group are kept together if possible, and other interactors are moved around to ensure this grouping can remain on the same screen.

The masks are then joined together in a Wizard-like format, so that the user can move back and forwards between them. The dialogue controller on the server waits until the user has completed the entire form, and then submits it to the web application as if it had been filled in on a single page.

DiaGen provides adaptation at both the dialogue and logical levels of the Arch.

A categorisation of these research tools according to the Plastic UI snowflake is in table 2.1.

A number of patterns emerge from this classification. Firstly, all the research tools attempt to plasticise the physical, logical and dialogue layers of the interface (in the terminology of the Arch model). The number of layers of the Arch at

	Components	Target
UIML	Up to dialogue	Platform/Some User (Internationalisation)
ARTStudio	Up to dialogue	Platform
TERESA	Up to dialogue	Platform
MARIAE	Up to dialogue	Platform
DiaGen	Up to dialogue	Platform

	Phases (design)	Phases (runtime)
UIML	Forward engineering	Toolkit
ARTStudio	Forward engineering	n/a
TERESA	Forward engineering	n/a
MARIAE	Forward engineering	n/a
DiaGen	n/a	Infrastructure

	UI Implementation	Actor (design)
UIML	Various	Human
ARTStudio	?	Human/System
TERESA	Various	Human/System
MARIAE	Various	Human/System
DiaGen	HTML	n/a

	Actor (runtime)	UI Computation
UIML	System	On-the-fly/Pre-computed
ARTStudio	n/a	Pre-computed
TERESA	n/a	Pre-computed
MARIAE	n/a	Pre-computed
DiaGen	System	On-the-fly

Table 2.1: Categorisation of research tools on the Plastic UI snowflake

which a tool can effect plasticity provides one way of quantifying the degree of plasticity it provides. The dialogue component is central and important. Thevenin et al. (2003) refer to it as the "keystone" of the Arch, and it mediates between the functional core of the application and the concrete controls that the user interacts with. Secondly, they all provide plasticity based on the platform that the interface is running on. UIML extends this by also providing limited plasticity based on the class of user (allowing for internationalisation).

There are two patterns, however, that do are not taken into account in the classification on the plastic user interface snowflake: and these need to be taken into account for the purposes of this thesis.

The first of these is that they share, to a greater or lesser extent, a very model-driven attitude towards user interfaces and towards the way that development

is done. ARTStudio, TERESA and MARIAE are explicitly model-driven engineering tools. DiaGen is not explicitly a model-driven engineering tool, but it is designed to fit in with an enterprise software development methodology that includes the use of the Unified Modelling Language ("UML") and the associated "Model-Driven Architecture" that the Object Management Group advocates (Miller and Mukerji, 2003). UIML is less model-driven than the others, sitting as it does at an uneasy intersection between a declarative notation and an imperative programming language (Phanouriou, 2000, ch. 3): but it still relies on the creation of a model of the platform on which it is running to provide adaptation.

The second is that they do not address the peculiar guideline-driven approach to usability that, as was noted earlier, is forced on developers for mobile applications. The literature about the tools does not engage with guideline-based approaches for creating applications, and this author's—admittedly subjective—attempts to use the tools for which software is available (TERESA, MARIAE and UIML) to produce guideline-compliant non-trivial mobile applications were a struggle.

2.7 Industrial tools

2.7.1 Mobile application development

In addition to the research tools outlined above, there is a large amount of interest in industry in developing "cross-platform" tools. These cross-platform tools aim to let developers "create applications for multiple platforms from almost the same codebase or from within the same design tool." (Jones et al., 2012, p. 11).

Jones et al., writing in a report published by Vision Mobile, listed one hundred such tools, and examined fifty-three in detail.

The survey below is based on forty-six of the fifty-three tools that Jones et al. surveyed. This is a large and rapidly-changing area, and any survey must necessarily be both partial and a snapshot in time: in the six months following the release of this report, one of the fifty-three had disappeared entirely; five had changed their name; and seven had changed ownership. Seven of the tools could not be included here:

Uxebo apparat.io has disappeared as of May 2013; its website at this time noted that its beta period was over, and neither software nor documentation was available.

Antenna AMP Studio, Antix Game Studio, iFactr Monocross, KonyOne and *webMethods Mobile Designer* are proprietary and neither software nor documentation is publically accessible.

UxPlus Aqua is only documented in Korean, which the author is unable to read. No documentation is available in translation, and the comments in the example source code included in the software development kit are also in Korean.

The Vision Mobile report divides the hundred tools into five distinct categories, based on their technical approach to the problem. These categories are:

JavaScript toolkits provide high-level capabilities embedded within an HTML renderer using JavaScript and other web technologies. They are agnostic about the mechanism which delivers the application to the device.

One prominent example of this kind of tool is Sencha Touch (described in Sencha, 2012). It provides both interface design and software architecture support. The former is implemented as a themable user interface library based on HTML 5; the latter is implemented as an extensive MVC framework. Other toolkits provide less complete functionality. jQuery Mobile (jQuery Team, 2013) provides only the user interface libraries and provides no support for software architecture at all. FeedHenry (FeedHenry, 2012) concentrates on providing abstractions for network communication.

App factories are tools which allow non-technical users to generate mobile web sites or applications by a graphical process. They do not generally require or allow the designer to write code.

One prominent example of this kind of tool is Wix Mobile (SutherlandGold Group, 2011). Since these tools are not aimed at developers, they are included here only for completeness.

Web-to-native wrappers are tools which enable applications written using web technologies to behave as if they are native applications running directly on a device. They also provide access to functions and sensors on the device that web applications would not have access to, such as capture of images from the cameras on the device and the ability to retrieve data from the user's address book.

One prominent example of this kind of tool is Adobe's PhoneGap (Adobe Systems, Inc., 2013). PhoneGap provides two basic functions. The first is a template native application for each platform which embeds that operating system's native web browser component and loads an initial page. The second is a library which allows the native application to intercept requests from the web application, to act on them, and to return values to the web application.

On top of these functions, PhoneGap provides a standard library which gives access to cameras, sensors, local storage, and communication tools.

Runtimes are tools which provide an abstraction layer or a virtual machine between an application and the underlying platform.

One prominent example of this kind of tool is RunRev's LiveCode. When an application that was built using this tool is turned into a mobile application which can be installed on a device, a small engine is embedded into the application. This engine provides the interpreter necessary to run scripts in the application, and the software necessary for the application to use hardware facilities of the device.

The distinction between runtimes and web-to-native wrappers is purely one of technological basis: web-to-native wrappers are runtimes which use web technologies. Beyond this statement, the Vision Mobile report gives no rationale for separating these two categories.

Code translators are a very general category, in which all the tools are placed which take either source code or compiled code and turn it into some other kind of source code or compiled code so that the original code can be run in a different context.

One prominent example of this kind of tools is XMLVM (Puder, 2005). This tool converts source code and bytecode to a variety of intermediate representations, provides transformations between these intermediate representations, and then converts these intermediate representations back to either source code or bytecode.

These five kinds of tool are classified according to the Plastic UI Snowflake in table 2.2.

This table, and all the subsequent analysis of the industrial tools in this section, comes with a significant caveat. The tools are classified based on *what they are intended to do*, rather than on a view of the whole of their features. This is important because a number of these tools are built using repurposed technologies, and have features from a number of different categories. An example of this can be found in the web-to-native wrappers: they are built using repurposed web technologies. Their intention is to provide a technical platform. However, because of their use of HTML and the technical requirements of the operating systems on which they run, they provide some functions which provide physical adaptation (such as HTML forms) and logical adaptation (such as the user interface for image capture).

	Components	Target
JavaScript frameworks	Various	Platform
App Factories	Various	Platform
Web-to-native wrappers	Platform	Platform
Runtimes	Various	Platform
Code translators	Various	Platform

	Phases (design)	Phases (runtime)
JavaScript frameworks	Forward engineering	Toolbox
App Factories	Forward engineering	Infrastructure
Web-to-native wrappers	Forward engineering	Infrastructure
Runtimes	Forward engineering	Infrastructure/Toolbox
Code translators	Forward engineering	Infrastructure

	UI Implementation	Actor (design)
JavaScript frameworks	HTML/JavaScript	Human/System
App Factories	HTML/JavaScript	Human/System
Web-to-native wrappers	HTML/JavaScript	Human/System
Runtimes	Various	Human/System
Code translators	Various	Human/System

	Actor (runtime)	UI Computation
JavaScript frameworks	System	On-the-fly
App Factories	System	On-the-fly
Web-to-native wrappers	System	On-the-fly
Runtimes	System	Pre-computed
Code translators	System	Pre-computed

Table 2.2: Categorisation of tool categories on the Plastic UI snowflake

Some patterns immediately emerge. All these tool categories are forward-engineering tools. All target the platform; that is to say, Jones et al.'s "cross-platform tools" are, in Thevenin et al.'s terms, "multi-platform tools".

One pattern that emerges from the classification in table 2.2 is that the level of adaptation on the Arch which the tool supports does not depend on its technical approach. Each individual tool, however, can be categorised according to the level or levels of the arch at which it performs adaptation. This categorisation of the forty-six tools is summarised in table 2.3. The full categorisation of each tool is in appendix A. Note that the total of the numbers in the table is more than forty-six: this is because several tools operate at more than one level of the Arch model.

The tools are heavily weighted towards physical presentation; there are some which perform logical presentation adaptation; but none which provide dialogue

Level	Number of tools
Dialogue	0
Logical Presentation	3
Physical Presentation	26
Platform	29

Table 2.3: Levels of plasticity in industrial tools. (see appendix A)

Architecture family	Number of tools
Architecture-neutral	20
MVC	7
Game-specific	7
Other	3
N/A	9

Table 2.4: Software architectures in industrial tools (see Appendix A)

adaptation. This does not, of course, mean that adapting the dialogue to different devices is impossible; many tools will allow code to retrieve information about the device and the form factor, which can be used with "if" statements to make simple adaptations. However, they do not provide adaptation of a centralised dialogue component, and any adaptation has to be done manually, in an ad-hoc manner, and ends up spread out in many places in the code.

One other important variable for understanding the nature of these tools, which is not mentioned by the Plastic UI snowflake, is whether they assume or provide support for any specific software architecture. A categorisation of the forty-six tools along these lines is summarised in table 2.4. The full categorisation is in appendix A.

Most of the tools are either architecture-neutral, and do not make any assumptions about software architecture at all (such as jQuery Mobile, which exists purely to put concrete components on the screen) or sit outside the entire concept of the software architecture of the system (such as the "App factories" and the source code translators). Seven of the tools used game-specific architectures. These vary greatly between the tools, but concentrate on tasks common in game development, such as putting sprites on the screen and managing events concerning in-game objects. The most popular single architecture family is MVC. Seven tools provide members of this architecture family. The remaining three tools that provide software architecture support each use a distinct architecture: one is a flowchart-based "visual programming" environment (Lunduke, 2012); one uses the Android "Activity" model (Google, Inc., 2013b); and one uses an architecture based on the Observer pattern (Erich et al., 1995; Balmer, 2011).

Notation type	Number of tools
Textual declarative	17
Graphical declarative	5
Direct manipulation	15

Table 2.5: Declarative notations in industrial cross-platform tools

There are no heavyweight model-driven engineering tools among the forty-six. IBM provides model-driven engineering tools in other contexts (such as the Rational product line), but their cross-platform tool itself is not a model-driven tool (IBM Corporation, 2013). Microsoft's Visual Studio can provide some support for model-driven engineering (Cook, 2007), but in place of building a model-driven tool on top of this for cross-platform mobile development, Microsoft instead assisted work on two existing non-model-driven cross-platform tools, PhoneGap and Sencha Touch (MacFadyen, 2012; Bansod, 2012). XMLVM is designed in a model-driven way, with models of file formats and CPU architectures, declarative transformations between them, and metamodels which describe the models, but this is its internal architecture, and not the way that developers who use the tool are expected to use it.

There being no model-driven engineering tools does not imply that there are no declarative domain-specific notations in use among industrial mobile development tools. Embarcadero's cross-platform design tools use a subset of UML, for data modelling; and as will be described below, Oracle's and Apple's development tools both provide a modelling language for the user's dialogue with the application. Some of the forty-six cross-platform tools also use declarative domain-specific notations, and these are categorised in table 2.5 according to the following types of notation:

Textual declarative notations are those which are edited as text. This includes notations which are embedded in HTML, XML or JavaScript Object Notation ("JSON").

Graphical declarative notations are those which are edited as graphics. This includes notations such as flowcharts or UML. This does not include direct manipulation interfaces where the developer directly edits a user interface.

Direct manipulation notations are those which very closely mirror the final user interface that the user sees. The developer effectively edits the final user interface.

This table demonstrates that declarative notations are by no means uncommon. Textual declarative languages are the most common. Graphical declarative

languages are not as common as textual languages but do exist. This means that a tool that uses an executable declarative language will not necessarily be unfamiliar to developers, so long as the nature of that language is familiar: so a tool that embodies case can use a declarative way of specifying other elements of the dialogue, so long as those declarative ways are familiar to the developer.

2.7.2 Other application-level industrial cross-platform tools

There is a class of industrial tools which provide solutions to cross-platform problems in a wider sense. Many of these emerged originally from the problem of building graphical user interfaces on computers running UNIX-based operating systems, where it is not feasible to make assumptions about the nature of the hardware upon which the application is running, or even the precise nature of the operating system. Some of these tools were either designed with mobile development in mind or were applied to mobile development some time after their original implementation, and these are summarised here.

The X Window System (Mansfield, 1993), often known as X or X11, is a windowing system, graphics framework and network protocol for an enormous number of operating systems and hardware platforms. It is extremely feature-rich, and a full description of what it does and what problems it solves would require a book-length treatment of its own.

Its application to mobile development has largely been to provide basic graphics support and basic windowing (thus sitting at the platform layer of the extended Arch model). In addition, the other toolkits outlined below were originally designed for X, and so making X available makes it considerably easier to make these other toolkits available.

X provides no adaptation above the platform level. While it is often possible to run an application designed for a desktop context which uses X on a mobile device which uses X, the very same user interface will be displayed on the mobile device as on the desktop computer (as in figure 2.6).

X implementations are used in mobile platforms derived from Maemo (including MeeGo, Mer and Sailfish OS), in some Sharp Zaurus software, in the OpenMoko smartphone platform and in Tizen.

GTK+ (Krause, 2007) is an interface toolkit which was originally created for the GIMP image editing application, but was then adopted by the GNOME desktop. In terms of the Arch model, it provides physical adaptation, being able to adapt the appearance of its components to fit in with different contexts (see figure 2.7). GTK+ originally used X to provide its platform level

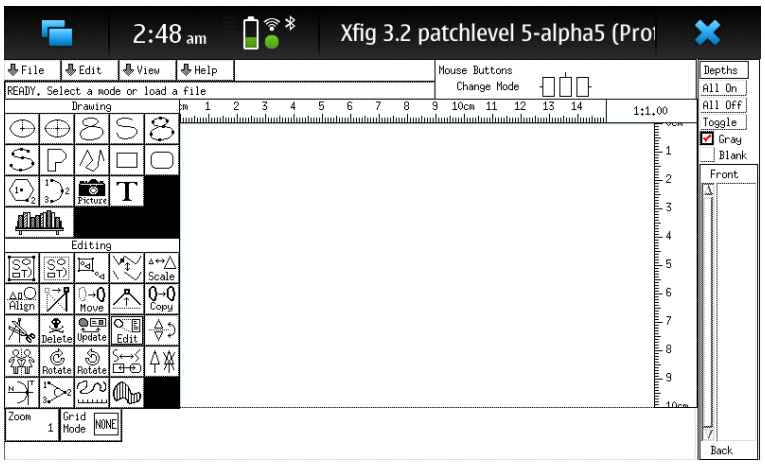


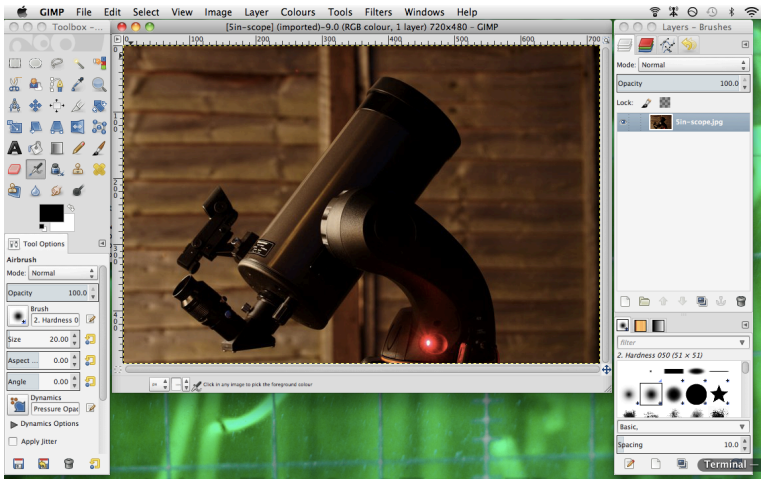
Figure 2.6: An unmodified X application running on Maemo 5

features, but more recent versions have interchangeable "backends" which allow it to use graphics frameworks other than X.

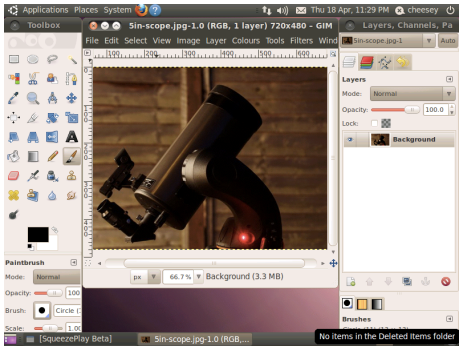
GTK+ was used in the Maemo mobile platform (although not on its descendents MeeGo or Sailfish OS, both of which use Qt) and the GPE palmtop platform. It continues to be used in the OpenMoko smartphone platform and the GNOME Mobile and Embedded initiative.

wxWidgets (Smart et al., 2006) is a meta-toolkit which provides a common programming interface over a variety of underlying user interface toolkits. It provides adaptation at the physical layer of the Arch model only. Its application to mobile development is in its availability on the iOS, PalmOS and Windows CE platforms, along with being available for a number of embedded platforms.

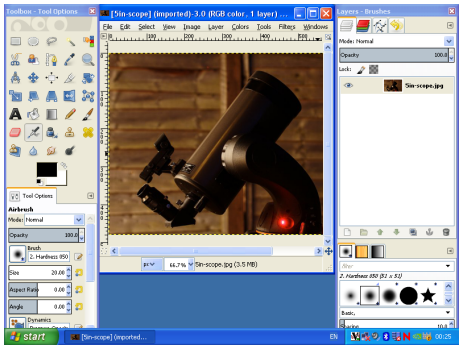
XUL is the Mozilla project's technology for building user interfaces for their applications. Unlike the toolkits above, XUL is declarative, with the structure of the interface being specified in an XML file. It provides adaptation at the physical layer of the Arch model. XUL is flexible, but proved to be slow and unwieldy on mobile devices (Raju and Duggi, 2008). Nevertheless while the Mobile Firefox browser was multi-platform (being available on both Android and Maemo) it used XUL: but when Nokia abandoned Maemo, Mozilla stopped maintaining Firefox for Maemo and Mobile Firefox effectively became a single-platform project available on Android only. In 2011, Mozilla announced that Firefox for Android would begin using a native, non-cross-platform interface, citing performance issues with XUL (Nightingale, 2011).



(a) MacOS X



(b) Ubuntu Linux



(c) Microsoft Windows XP

Figure 2.7: A GTK+ application on multiple desktop platforms

Physical adaptation can be best seen in scroll bars and drop-down menus.

	Soft. Components	Target
X11	Platform	Platform
GTK+	Physical	Platform
wxWidgets	Physical	Platform
XUL	Physical	Platform

	Dev. Phases (design)	Dev. Phases (runtime)
X11	n/a	Infrastructure/Toolbox
GTK+	Forward engineering	Infrastructure/Toolbox
wxWidgets	Forward engineering	Infrastructure/Toolbox
XUL	Forward engineering	Infrastructure/Toolbox

	UI Implementation	Actor (design)
X11	Various	Human
GTK+	Various	Human/System
wxWidgets	Various	Human/System
XUL	Various	Human/System

	Actor (runtime)	UI Computation	UI Migration
X11	System	On-the-fly	Possible with effort
GTK+	System	On-the-fly	n/a
wxWidgets	System	Pre-computed	n/a
XUL	System	On-the-fly	n/a

Table 2.6: Categorisation of non-mobile tool categories on the Plastic UI snowflake

These tools are summarised in table 2.6. They are very similar to the mobile-specific development tools: each targets the platform area, and they adapt at the physical layer of the Arch. The only departure is that X11 is the only tool studied here—including the research tools—that allows the *running* interfaces to be moved between devices.

2.7.3 Web standards and design approaches

Perhaps unsurprisingly, much of the industrial work on building plastic user interfaces comes from the building of web applications. In these applications, the actual application is largely implemented on a web server, and the interface is served to the client using web protocols.

The web's lifeblood is not individual tools, but rather standards and design approaches. There is no longer, for example, any reference implementation of web page rendering: instead, the method by which it is to be done is laid out in de-

tail in a standards document. Software developers who design and build rendering engines adhere—to a greater or lesser extent—to these standards. Web and Internet standards are generally laid out in a single canonical reference document. Examples of web standards are the documents published by the World Wide Web Consortium.

Design approaches are less formal than standards: they do not necessarily possess a canonical document, but rather they emerge from the community's attempts to solve problems. A design approach is at best nebulous, and will usually consist of some articles or documents demonstrating the usefulness of the approach, its goals and its approximate methods, and an array of example implementations.

Here follow some characteristics of the major standards in use.

Media queries are a feature of the Cascading Style Sheets ("CSS") standard which allows different styles to be applied to elements of an HTML document based on characteristics of the display on which they are being presented, and thus to present different versions of an interface or a design depending on these characteristics.

CSS version 2 introduced a feature called "media types" (Bos et al., 2011) that allows different styling to be applied based on the type of display which is in use. The standard mandates a list of types of displays which must be distinguishable in stylesheets. These types are:

- Letter or line-at-a-time braille devices.
- Page-at-a-time braille devices or braille printers.
- Handheld devices with small screens and limited bandwidth.
- Printers.
- Projected presentations.
- Computer screens.
- Speech synthesisers.
- Devices with a fixed-width character grid and no graphics capability (such as text-only terminals).
- Television screens.

CSS files consist of a list of rules which map "selectors" to styling instructions. Selectors are predicates which identify a subset of the HTML elements within the document. When a rule is applied to the document, any elements that match the selector are styled according to the styling instructions in the rules. Media types are applied to sets of rules. Any rule to

which a media type is applied will only affect the document if the media type matches the display mechanism which is actually in use.

Media queries (Rivoal et al., 2012) were introduced in CSS version 3 in order to make this mechanism more fine-grained . In CSS version 3, the media type applied to a block of rules can be refined by a set of criteria defining more precisely when this rule should be applied. Each criterion consists of a characteristic of the display device and either a range in which the value associated with this characteristic must fall, or an exact value to which the characteristic must be equal. The characteristics that CSS version 3 supports are:

- The width and height of the area of the display device that is being used to display the page.
- The width and height of the whole display device that is being used to display the page.
- The orientation of the display device that is being used to display the page (portrait or landscape).
- The aspect ratio of the area of the display device that is being used to display the page.
- The aspect ratio of the whole display device that is being used to display the page.
- The number of colours available to the document.
- The resolution (pixel density) of the device that is being used to display the page.
- The kind of scan (progressive or interlaced) of the television-type device that is being used to display the page.
- Whether the display being used to display the page is based on a fixed-width single-font character grid or whether it can display multiple fonts and graphics.

Note that this does not permit *parameterisation* of the rules by these variables: it is not possible, for example, to use the aspect ratio of the display in some arithmetic way. They purely enable and disable rules.

Note also that the characteristics are all properties of the display itself. Using CSS version 3, it is not possible to enable and disable styling rules based on the types of input device in use. This means, for example, that it is impossible to use media queries to increase the size of hyperlinks on touch screens compared to screens which are paired with a mouse.

Since CSS is generally concerned with the appearance of interactors, CSS version 3 media queries provide adaptation at the physical layer of the Arch model.

Indie UI is a standardisation project being pursued at the World Wide Web Consortium's Web Accessibility Initiative. The Indie UI project is currently still in its very early stages and no standards have yet been published. Therefore, this section is based on the working group's drafts as available in May 2013 (Craig and Cooper, 2013; Craig, 2013). The "User Context" specification was especially fluid at this time.

Indie UI is a "a way for user actions to be communicated to web applications" so that web applications can "work in a wide range of contexts — different devices, different assistive technologies (AT), different user needs." (Henry, 2013). It provides specifications for two scripting application programming interfaces that can be used by web applications.

The first is a means for a web application to query the user's context, and to gather information about what extra accessibility needs the user may have (Craig, 2013). These are gathered into a number of groups. These are:

General settings. These allow the developer to find out whether the user needs to be able to use the application using only the keyboard, and whether they have inverted the colours on their screen.

Type settings. These allow the developer to find out what sizes of text the user needs, whether they need custom letter or word spacing and whether they need a custom line height.

Display settings. These allow the developer to find out whether the user needs increased contrast on their display, whether they need content to be displayed only in greyscale and whether they need text to be displayed in a light colour on a dark background.

Media alternative settings. These allow the developer to find out whether the user needs subtitles for any video that the application might display, what language those subtitles should be displayed in if they are needed, what type of subtitles to use (whether to use specific subtitles for the hard of hearing or simply to transcribe the spoken content), whether the user needs a transcript of any video that the application might display, whether the user prefers to be presented with a video of sign language where it is available, and whether the user needs an audible description of content presented visually.

Screen reader settings. These allow the developer to find out whether the user is using a screen reader, and if so which screen reader is in use.

The rationale for the choice of these categories and these accessibility settings is unclear as of May 2013, as the explanatory text around the specification of the programming interface has yet to be written.

The second tool is an extension of the way that user interface events are delivered to web applications. In standard JavaScript and HTML, many events directly represent users' use of input devices. For example, elements of an HTML page can receive "click" events when the user clicks on them with a mouse; "touchdown" events when the user puts their finger on them using a touch screen; and "keydown" when the user presses a key when they have input focus. There are other events which represent the user's intent, and which are decoupled from the specific user input. For example, elements of HTML page can receive "copy" events when they are copied to the user's clipboard to be pasted into another document.

Indie UI's events framework provides further events, which represent the user's intent. This is so that web browsers used by users with accessibility needs can provide appropriate interaction methods for the user to signal these intents without the web application needing to be aware of what these interaction methods are. It also allows web browsers for users with different kinds of device to let those users use appropriate idioms for their devices. The Indie UI event framework provides events for users' intents to undo the previous action; redo the last undone action; expand a collapsed section of the page; collapse an expanded section of the page; dismiss a popup or dialogue; delete an element on the page; move an element of the page; pan an element on the page; rotate an element on the page; scroll the page; and zoom in or out.

How the user context specification and the events specification will interact is not yet clear.

Since the Indie UI events framework abstracts the web application away from individual interactors and towards the user's intent when they use the interactors, it provides adaptation on the logical level of the Arch model.

Responsive web design is a design approach introduced by Ethan Marcotte (2010a) to allow a designer "to use the same code to deliver an experience to desktop browsers, tablets and mobile devices" (Bryant and Jones, 2012, p. 37). Since the word "experience" as used in the web design community certainly includes the concept of usability, it is reasonable to examine this approach as a way of building plastic interfaces.

Responsive web design consists of three techniques used together which complement each other.

The first of these techniques is the use of a "fluid grid". Web design inherited from print-based design the concept of a "typographic grid". A typographic grid is a regular grid which is laid over the page on which the designer is laying out text. Text and images are laid out on this grid. The regularity of the grid ensures that there are visual rhythms in the page that the eye can easily follow. In print-based design, the size of the grid is expressed in absolute units (such as millimetres, inches or points). Since the size of the paper is unlikely to change once it has been printed, this does not put undue limitations on the kinds of design that the grid can accommodate. When this technique is used on the web, however, where the size of the display medium can change at any time, it leads to fixed-size designs which cannot resize to fit the space available to them. A fluid grid is a typographic grid where the dimensions of the grid, instead of being expressed in absolute units, are expressed in units which are proportional to the size of the display medium (such as percentages) or to the size of the text in use (such as ems) (Marcotte, 2009). This allows the grid to react to changes in the size of the display medium without losing its regularity.

The second technique is the use of "fluid images". Pixel-based images, such as photographs or textures, have a defined size in pixels. Pixels are an absolute unit of measurement: the size of a pixel may vary from screen to screen, but it does not vary depending on the size of the display medium in the same way that a percentage or an em might. Fluid images have a maximum size specified in the same proportional units as the fluid grid of which they are a part (Marcotte, 2010b). This prevents them from spilling over the edges of the design element which contains them.

The third technique is the use of CSS version 3 media queries to allow the fluid grid to be rearranged to fit the size of the display medium and to hide unused functionality. Columns of the grid may be dropped, so a three-column layout may become a two-column layout at a smaller size; horizontal arrangement may give way to vertical arrangement, so a navigation bar which is a vertical list of links on the left hand side of a large display medium may become a horizontal list of links at the top of a small display medium; and less important content may be dropped entirely, so that infrequently used navigation links may only be visible on larger display media. Wroblewski (2012) has identified a number of ways in which the grid has been successfully adapted to smaller display media.

One high profile use of the responsive web design approach has been the web site of the Boston Globe newspaper. This web site can be seen at three different sizes of display medium in figure 2.8. All three techniques can be

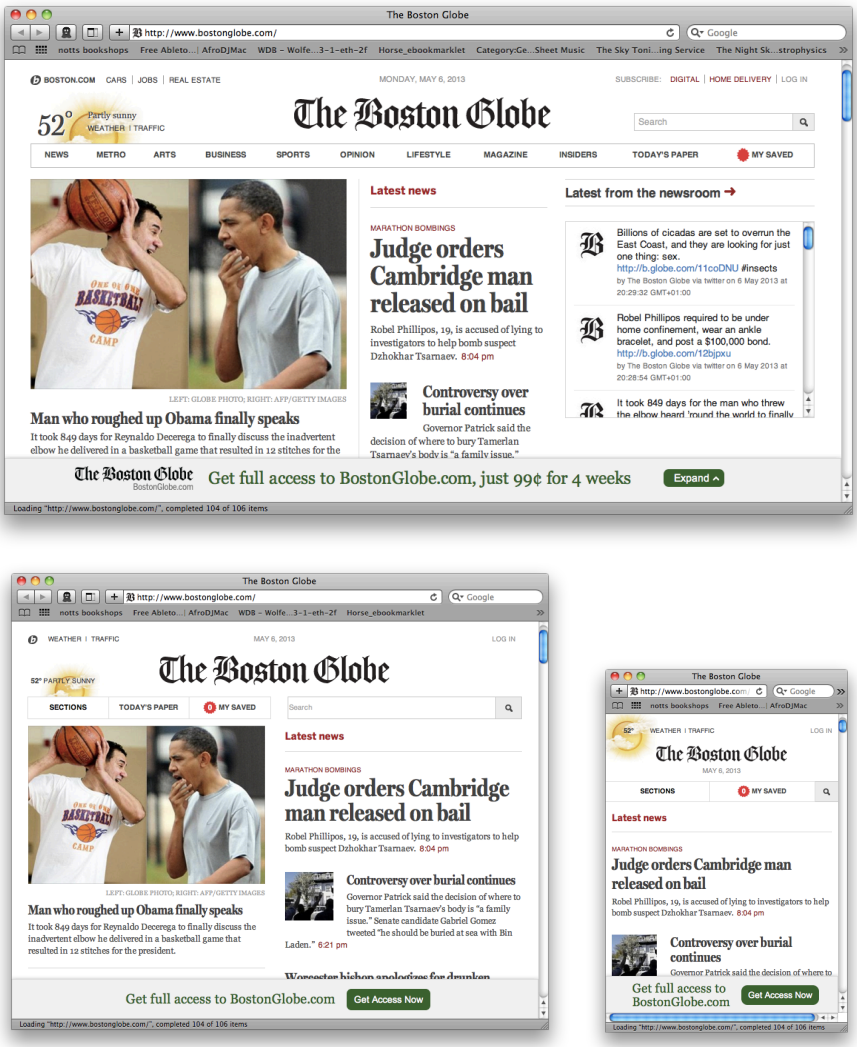


Figure 2.8: The Boston Globe website at three sizes

seen operating in this web site. The fluid grid allows the columns of information to change size as the display medium changes size. The image illustrating one of the news stories changes size in response to the changes in the grid. The number of columns in the grid changes downwards as the display medium size decreases, and unimportant content disappears.

Responsive web design operates at the physical layer of the Arch model, as it adapts how things appear based on the size of the display medium.

Adaptive web design is a term which was introduced by Gustafson (2011) to refer to the use of a "progressive enhancement" approach along with the use of

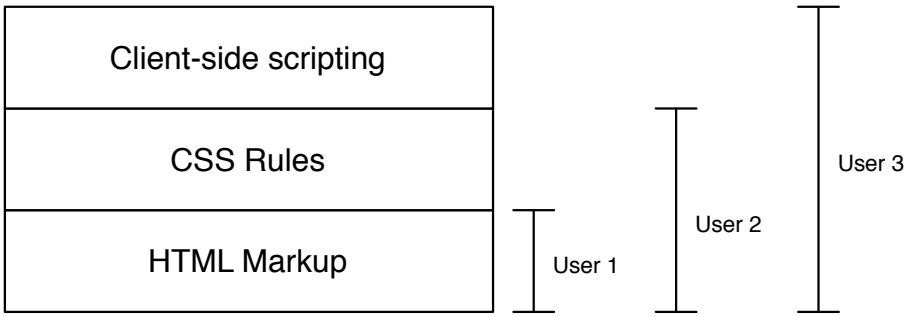


Figure 2.9: Layers of progressive enhancement

multiple, fixed-width designs which are switched between using CSS media queries.

Progressive enhancement is a technique which aims to provide functional and usable web interfaces to various devices by considering the design as a layered entity (illustrated in figure 2.9). The bottom layer consists of simple, semantic HTML markup. The second layer consists of the CSS styling rules which are applied to the HTML. The third layer consists of client-side scripting for design elements which cannot be implemented using CSS. Each layer may only depend on the layers below it for functionality: a browser which cannot display CSS or run JavaScript must still be able to provide a usable and functional experience using only the HTML markup (Wells and Draganova, 2007).

2.8 Structuring the dialogue component

Each of the research tools above uses a domain-specific language to allow the developer to specify the dialogue component. The industrial tools do not permit adaptation at the dialogue level, but some still have domain-specific languages to specify the dialogue component. These languages, both from the research and industrial tools, are summarised here.

ConcurTaskTrees ("CTT") are a notation for task structure and dialogue structure which concentrates on providing a hierarchical structure based on the activities that the user is attempting to perform (Paternò et al., 1997).

A ConcurTaskTree consists of two sets of relationships: firstly, a hierarchical relationship between tasks and subtasks, which is represented as a vertical tree where each task and subtask is represented by a node; and secondly a set of temporal relationships between subtasks at the same level, which are represented as edges between nodes at the same level. Each node carries a

rich set of metadata, which specify whether the node represents an action to be carried out by the user or the computer, what kind of action it represents (such as "editing" or "monitoring"), how often it is likely to be performed, whether it has any preconditions and whether or not it is an optional action.

The hierarchical relationship between two nodes, referred to here as *parent* and *child* is one of containment or necessity. Paternò (2003) describe it as "in order to [perform the parent task] I have to [perform the child task]" (p. 490).

The relationships between nodes on the same level are richer, and can represent a number of different ways that the nodes depend on each other. The relationships that the CTT notation can encode are:

Enabling: The second task cannot begin until the first has been performed. For example, in a satellite navigation application, the system may not attempt to find the user's GPS location until the user has requested their current position.

Choice: Only one of the two tasks can be performed: once one has been started, the other cannot be performed. For example, in a satellite navigation application the user may have a choice either to view points of interest near them, or search for points of interest by name.

Enabling with information passing: The second task cannot begin until the first has been performed; information from the first task is passed to the second. For example, in a satellite navigation application, the computer cannot pan the map to the user's current location until it has located the user through GPS; and the information gathered from GPS is used to pan the map to the correct place.

Concurrent tasks: The tasks can be performed in any order, or at the same time. This includes the possibility of being able to interrupt the first task to switch to the second, or interrupt the second to switch to the first, as many times as the user wishes. For example, in a satellite navigation application, if the user is recording their current route, they can pan the map around without interrupting or affecting the recording of the route.

Concurrent communicating tasks: Concurrent tasks which exchange information with each other. For example, in a satellite navigation application, the user can be panning the map around and measuring distances at the same time. Information flows from the map panning task to the distance measuring task. If it did not, then if the user started

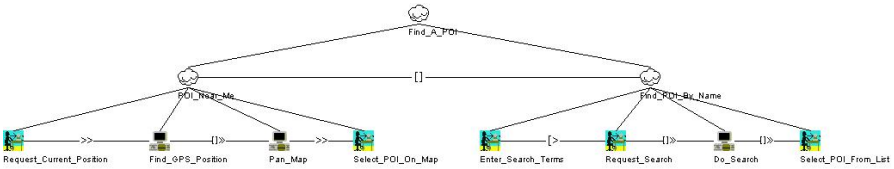


Figure 2.10: An example of CTT notation

measuring a distance then panned the map, then the distance measuring task would have no way of knowing where the far end of the line was on the map.

Independence: The tasks can be performed in either order, but once one has been started, it must be finished before the other task can be started. For example, in a satellite navigation application, a search task may have this relationship with the tasks that can be performed on the map, such as panning and distance measurement: there is no specified order, but once the search task has been started it takes up the entire screen and the user cannot continue performing actions on the map until the search has been completed or cancelled.

Disabling: The first task is interrupted permanently by the second task. For example, in a satellite navigation application, when the user is searching for a point of interest, their entering of data in the search form is interrupted by their pressing of the "Search" button.

Suspend-resume: The first task can be interrupted by the second; once the second has been completed then the first can be resumed from where it was left off. For example, if a satellite navigation application had an option to email details of a point of interest to another person, then the process of viewing the point of interest's details would be suspended by choosing to send the email, and entering the email address to send it to. Once the entering of the address was complete, then the viewing of point of interest details could resume from exactly where it left off.

An example of a task structure in CTT notation, following several of the examples above, is in figure 2.10.

The CTT notation is used in the ARTStudio, MARIAE and TERESA tools outlined above.

State networks are a family of notations which share the idea that the system they describe can be in one of a fixed set of states, and that user input makes the system change from state to state.

In the context of human-computer interaction, states in a state network correspond to Berti et al. (2005)'s presentations, with the paths that the user can take out of the state corresponding to the options they have for interacting with the system while the system is in that state.

State networks are somewhat controversial in use in the area of human-computer interaction. In his thesis, which described UIML, Phanouriou (2000) suggested that they are not useful because "most modern user interfaces are mode-less (the system can be in more than one state at any time)" (p. 15) while state networks permit the system to be in one state alone. However, interactions with a mobile device are modal in a way that interactions with a desktop are not: Apple's user interaction guidelines for iOS do not allow elements of multiple different presentations to be on the screen at once (Apple, Inc., 2012a).

A number of variations on state networks have been used to describe interfaces. Facebook's "SproutCore" JavaScript toolkit uses Harel (1987)'s state charts, which are state networks which can be nested inside one another, to describe the dialogue layer in mobile applications (Sarnacki et al., 2012). Thimbleby (2007) also used state charts to define the behaviours of systems; and (Thimbleby et al., 2011) defined a type of state network which had a conception of numeric "buffers" to store extra information about the state of the system.

Dialog Flow Notation ("DFN") is a modified form of state network used by Book and Gruhn (2004). It is used as part of their Dialog Control Framework, which is a tool for building web applications using a separate dialogue controller.

DFN diagrams consist of a set of nodes with edges connecting them all contained within a box (a "contour") that contains the whole diagram. A key to this notation is in figure 2.11. There are two elementary kinds of node: a hypertext page that is presented to the user and a piece of logic that is performed by the application. Two special types of node represent possible entry points into the state network: the "initial event" type represents the point at which the user begins the dialogue, and the "abort event" (which may not be present) represents a point to begin the dialogue should the user choose to abort the dialogue entirely while engaged in it. This abort event may, for example, lead to an area of the dialogue in which the user is prompted to save their changes.

There are three extra kinds of node which are used to embed one diagram within another. In a diagram that is embedded in another,"terminal events"

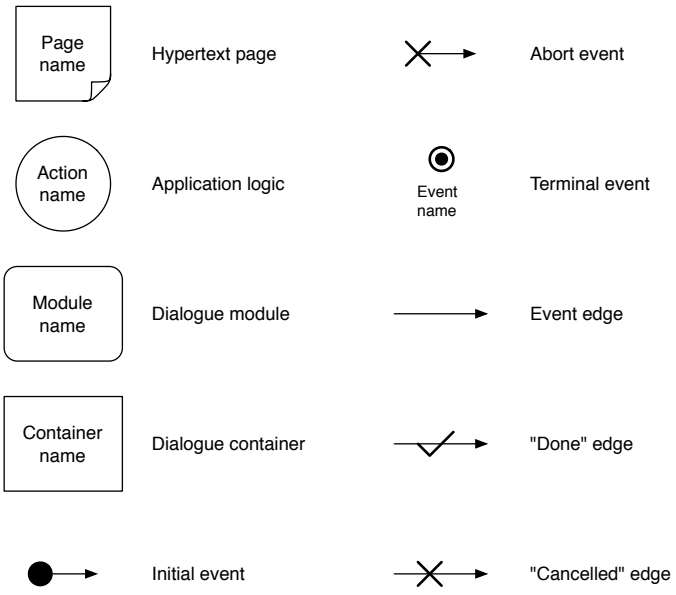


Figure 2.11: A key to the DFN notation

are nodes which return an event to a parent diagram. In the parent diagram, the child graph is represented as a "dialogue module" node if it generates events or a "dialogue container" node if it does not. A dialogue module must have an outgoing event edge for each terminal event node in the diagram it represents.

Edges between the nodes in a dialogue diagram are labelled with "event names", which are arbitrary strings. Events which emerge from a hypertext page or from an application logic node are defined by the implementation of that node. Events which emerge from a dialogue module are defined by the terminal event nodes in the graph that the dialogue module node represents. There are convenience notations for events which represent the completion of a child graph or which represent the user choosing a "cancel" option.

An example of a dialogue specified in the DFN notation is shown in figure 2.12. This figure shows part of a dialogue for a satellite navigation system. This illustrates the features of the DFN: application logic and hypertext being presented to the user are both represented; the two diagrams are nested, with the "Search or filter POIs" node in the lower graph corresponding to the entire upper graph; the upper graph returns events to the lower graph; and there are shorthand notations for common kinds of edges.

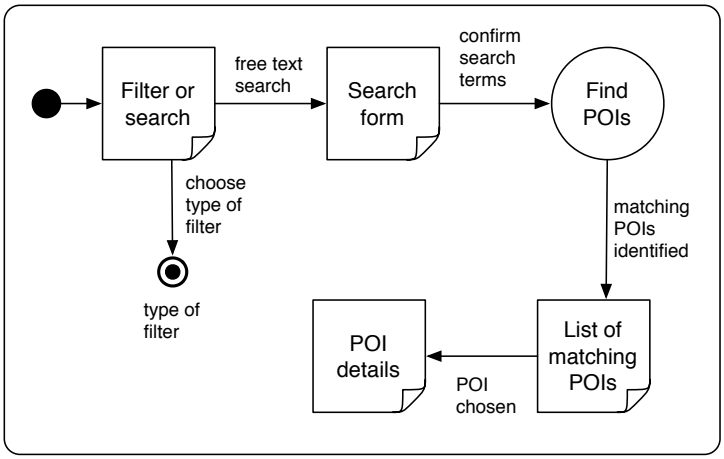


Figure 2.12: An example of the DFN notation

The DFN is used in Book and Gruhn's Dialog Control Framework, which forms the basis of the DiaGen tool.

Storyboards are a modified form of state network used in Apple's recent development tools for iOS (Apple, Inc., 2012b). They consist of a directed graph of nodes. Each node is either a view with its corresponding controller or a standalone controller. One of these nodes is marked as an initial node.

The graph has three kinds of edges. The first kind goes from a "from" node to a "to" screen and represents a visual transition after which the screen is displayed. The second kind goes from a "from" node to an action in a controller, and represents a call to that controller action. Each of these two kinds of edge is labelled with an event which can be triggered by the node from which the edge emerges. When this event happens (for example, a user touch), the edge is followed, either to another screen, or to the controller action which is then performed. The third kind of edge in the graph goes from a screen and its controller to another view, and represents containment: for example, a screen with a tab bar in it would have this kind of edge leading from it to as many other screens as there are tabs in the bar. These screens would be visually contained by and managed by the view containing the tab bar and its corresponding controller.

An example storyboard is given in figure 2.13. The initial node is a "Navigation controller", which manages a history stack and provides a "back" button when the stack contains somewhere to go back to. The icon on the edge that links this controller with the main screen indicates that this is a management or containment edge. The "Details" screen is linked to the main screen by an edge which is labelled with a touch event on the MapView component; the "Current Location" view is linked to the main screen by an edge which is labelled with a touch even on the "Current" button.

At run time, each event from the current view is passed to the storyboard, rather than directly to the view's controller; the storyboard then is either responsible for calling a controller action, or for displaying a new view. There is only one storyboard, regardless of how many views and controllers there are. The iOS MVC architecture with a storyboard is illustrated in figure 2.14.

Netbeans J2ME Flow diagrams are a specialised form of state network provided by Oracle's NetBeans J2ME "flow designer" (Keegan et al., 2006). This allows the developer to edit a directed graph of nodes. The nodes consist of screens, methods to be performed, named "entry points" at which the dialogue can be started, and a special "Mobile device" node which represents

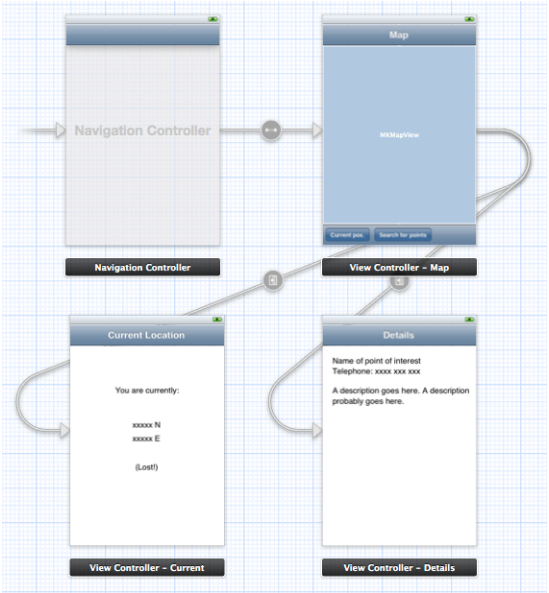


Figure 2.13: A storyboard

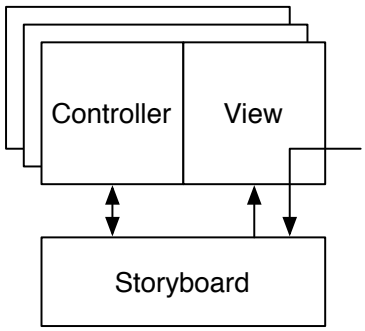


Figure 2.14: The Cocoa "Storyboard" mechanism

the operating system outside the application, and functions as a start and end node (see figure 2.15).

Edges are labelled with an event which can happen at their starting node, with a maximum of one edge per event; what these events are varies depending on what kind of node they emerge from. From forms, for example, the allowable events are J2ME command objects (JSR271 Expert Group, 2009); from the "Mobile device" node, the allowable events are the application being started and the application being resumed from the background. Events are generated by the screens and are fed to the dialogue component, which then calls actions in the underlying user code, and changes which screen is displayed. This is illustrated in figure 2.16.

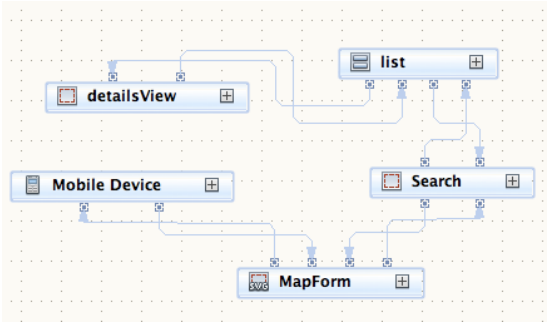


Figure 2.15: A NetBeans "Flow"

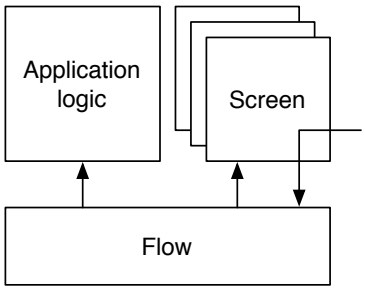


Figure 2.16: The NetBeans "Flow" mechanism

Card-based approaches are those approaches in which the interface is considered as a stack of cards which contain information. All the cards are the same size, and only one card at a time is visible. This approach was introduced with Apple's HyperCard tool in 1987. As of mid-2013 RunRev's LiveCode (RunRev Ltd., 2010) is the only tool which uses this approach for building mobile applications.

An application's interface is referred to as a "stack". A stack consists of an ordered list of cards and a set of backgrounds. Cards and backgrounds can both contain graphics, editable text, hyperlinks and buttons. Additionally, every card belongs to one background, and the objects which are on that background are displayed on the card, behind the card's own content (Apple, Inc., 1990).

These concepts are illustrated in figure 2.17, which shows two cards from the "Home" stack from HyperCard 1.2. The icons, time and card title on the first card, and the text fields, radio buttons and card title on the second card are on the card itself. The back and forward buttons at the bottom and the graphic of the stack of cards are on the background, and are shared between all cards which belong to that background.

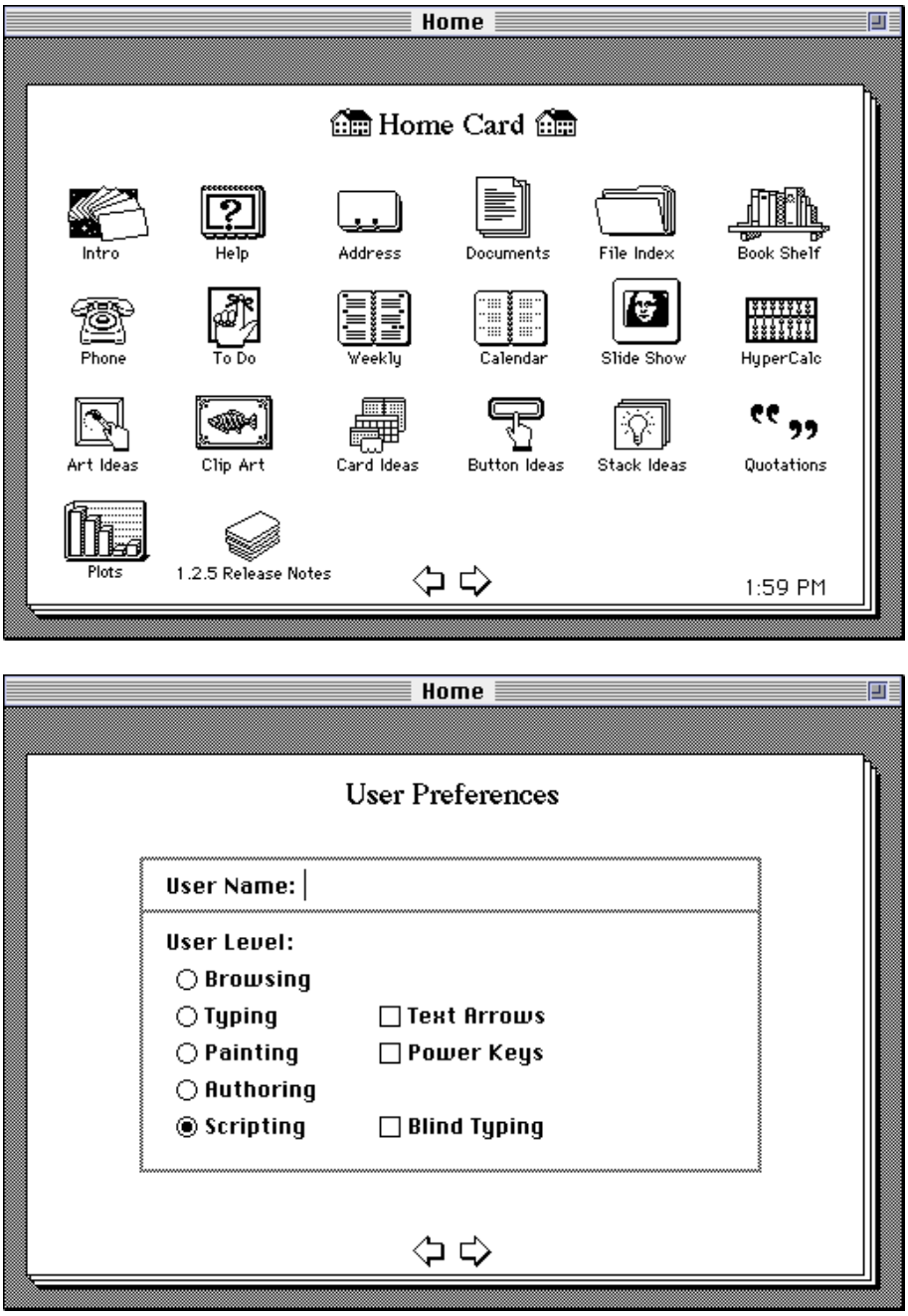


Figure 2.17: Two cards from the HyperCard 1.2 “Home” stack.

Card-based approaches are a lot like state network based approaches: the system can be in a single state, or showing a single card at once, and on certain user interface events, the system moves between one state or card to another state or card. They are not, however, the same: and the reason for this lies in the way that card-based applications store data.

In a stack, each card stores the data that is entered on it. Users will tend to create new cards in stacks as they enter data. For example, in the case of an address book application, the application would provide a background with the appropriate fields for name, address, telephone number, and so forth. This background would also provide a button that would create a new card belonging to this background. In the resulting stack, each address stored would be on its own card, so the designer cannot know at design-time how many cards the stack will contain. By contrast, the designer of a state machine does know at design-time how many states the system can be in.

Note that, following Java, both Sencha Touch and jQuery Mobile (Oracle Corporation, 2013; Sencha, 2012; jQuery Team, 2013) provide a CardLayout facility. This is a collision in terminology. These facilities are layout managers which permit only one component to be visible at a time. They do not implement a card-based approach as implemented in LiveCode or HyperCard.

2.9 Summary and conclusions

The research tools available provide a high degree of plasticity, allowing for adaptation up to the dialogue level of the Arch model. However, comparing them to the tools that the industry has generated for itself reveals two major ways that they do not address the smartphone developer's conundrum. These two ways correspond to the two patterns that were identified that were not within the plastic UI snowflake.

Firstly, they use very different software frameworks and conceptual frameworks to the ones that mobile developers use every day. An examination of the industrial tools shows either that those tools make no assumptions about what software architecture is in use or use one that is not heavily model-driven. This is unlikely to be through ignorance: as noted above, IBM and Microsoft have both developed model-driven tooling but chose not to use this in their cross-platform mobile tools. Taking large companies to be monolithic is inadvisable, as there is no guarantee that the left hand knows what the right is doing. However, IBM, at least, provide extensive documentation for integrating applications built with their cross-platform toolkit into systems built using their enterprise development

tools, which do include model-driven tools. Not only, then, is there no evidence of model-driven tooling in the industrial cross-platform mobile development tools, but the organisations that one might have expected to use model-driven tooling in their cross-platform mobile development tools have not done so.

It is important to note here that this does not mean that there is any objection among these tools to modelling or to declarative notations in general; a number of declarative models in use in industrial tools were given above for dialogue. Evidence was also given above that these industrial tools use notations for other things besides dialogue. What is not present, however, is the explicit model-and-transformation approach that makes up the core of model-driven engineering and that explicitly drives the research tools.

Secondly, the research tools do not follow a guideline-based approach to usability. The goal of the industrial tools is generally a very pragmatic one: to let a developer create a mobile application that they will actually be able to deploy and sell. The physical and logical adaptation that they provide is designed to be sufficient to produce applications that are close enough to the platform guidelines that they can be distributed. The research tools do not have this focus, and do not engage with a guideline-based approach to interface design.

The industrial tools and standards, while geared to the particular definition of usability required by mobile developers, have significantly limited expressiveness compared to the research tools. Every research tool surveyed can effect plasticity, at least to some extent, up to the dialogue level of the Arch model. None of the industrial tools can. A number of them have a notation or a declarative concept of dialogue, but none of them effect plasticity through it.

Therefore, there is a gap in expressiveness between the industrial tools and the research tools, and that gap is defined by a number of simple requirements. A tool or model that bridges this gap should provide a useful degree of plasticity at the dialogue level, should make use of pragmatic methods that are familiar to developers, and should embrace a guideline-based approach to usability. The next chapter suggests that the notion of "case" from natural languages can help to bridge that gap.

Chapter 3

Case and its application

3.1 Introduction

Chapter 2 demonstrated the need for plastic user interfaces among mobile developers and introduced the state of the art in tools used to create applications with plastic user interfaces. It also surveyed the ways that these tools manage the dialogue between user and machine. The dialogue between the user and machine is a problem of ordering: users do things in an order that is mapped to a task structure. Finally, the chapter concluded that there was a gap in the tooling available.

This chapter suggests that the notion of "case" from natural language might be helpful in bridging this tooling gap. Case is a mechanism that is common in natural languages and which permits flexible word order (analogous to flexible ordering of elements in a task in a plastic user interface) while retaining semantic clarity. Furthermore, the chapter presents some of the evidence that lets linguists argue that this is a genuine pattern across languages and not just a chance resemblance. Its commonness suggests that case may be linked to some deeper phenomenon that is innate to people's ability to think symbolically, and this could be a natural and intuitive notion for structuring user interfaces. The discussion of case in the strict, linguistic sense in this chapter forms a justification for even considering the use of case in plastic user interfaces.

This chapter starts by explaining what case is and how it appears in natural languages. In English, sentences have a strict word order; but in other languages, the word order can change depending on the demands of the context in which the language is being used. Case provides a major mechanism by which this process of change can happen. Latin provides very good examples of this. As an illustration this chapter takes two examples from Latin literature to demonstrate how case can function to produce flexible word order, and ways in which this flexible word order can be used.

If case only occurred in Latin, applying it as a concept outside of Latin and its descendent languages would be dubious. This thesis proposes to apply it out-

side of language altogether. Therefore, to demonstrate that case is widespread, the chapter presents a small illustrative survey of languages that use case. For the purposes of this chapter, twenty languages are chosen from a wide range of language families and a wide range of time periods. This makes it unlikely that all the case systems are borrowings: some of them at least must reflect a cross-linguistic phenomenon.

After this, the chapter draws out some common meanings from the case systems of languages. Four examples of common patterns are given, which are the patterns that will be engaged with during the remainder of the thesis. These patterns are the top four items in Blake's (2001) case hierarchy, which is described next. Blake's case hierarchy predicts the structures of case systems found in natural languages, and does so reliably: it is upon Blake's case hierarchy that the choice of the cases to be used in the plastic user interface framework will be based.

The application of case to plastic user interfaces is discussed in general terms. Two potential reasons why case might be useful are summarised, a simple example is given of different ordering assumptions being used on different platforms, and examples are drawn from user interface guidelines to demonstrate that the meanings of cases summarised in this chapter have analogies in user interfaces. These ideas will be carried forward into the remainder of the thesis. Finally the research questions that the remainder of this thesis will answer are presented in the light of the information provided in this chapter and in chapter 2.

3.2 What is case?

Case is a means of categorising the relationship between pairs of words; specifically, as Blake (2001) puts it, "between a noun and its head".

The head of a phrase is the "dominant" word in the phrase; it is the word that carries the fundamental meaning, and without which the phrase does not make sense. The other elements in the phrase or compound modify or clarify one's understanding of the head, and are called "dependents". If one were to draw the phrase out as a tree, then the head would be at the top of the tree.

Example: In the phrase "the big dog", the head is "dog"; and the phrase is a noun phrase, because the head is a noun.

In some circumstances, for example, in attaching a set of adjectives to a noun, each of the adjectives modify their head in the same way. In the phrase "the circular purple dog", "circular" and "purple" indicate orthogonal (if slightly unnatural) properties of the dog. Reordering the phrase as "the purple circular dog" leaves the animal in exactly the same state. Likewise, attaching a set of adverbs to an

adjective or a verb is generally symmetrical. In the former case, the head of the phrase is a noun; in the latter case, an adjective or verb.

This symmetry breaks down when attaching nouns to other words; most of the time, these nouns are not interchangeable or symmetrical with regard to their head at all.

Example: The sentences "the dog bit the boy" and "the boy bit the dog" are not in any way equivalent (with respect to the events they are describing).

Example: The sentences "the girl gave the dog to the boy" and "the girl gave the boy to the dog" are not in any way equivalent.

This means that the relationship between the noun and its head has a meaning distinct from the meaning of the words themselves. Case refers to the categorisation of these relationships in the way that the language is used. Languages that use case have a set of cases each of which specifies which relationship is meant. These categories are usually indicated by a modification of the noun. The English pronoun "he", for example, exhibits three cases, indicated by the three forms "he", "him" and "his".

Some of these categories are strictly grammatical, but they need not be. Examples of the former include languages which put the subject and object of a transitive verb in different cases. A transitive verb is one which takes a subject and an object. In English, a good example is "bites" ("the dog bites the man"). An intransitive verb takes only a single noun and no object - in English, a good example is "falls" ("the boy falls"). Some verbs can occur in both patterns - an example being "feeds" ("the cow feeds"; "the girl feeds the dog").

This pattern, where the subject and the object are in different cases, can be seen in English pronouns:

Example: "she hit him" against "he hit her".

Other cases do not fulfil purely grammatical roles, as these do, but encode a semantic or locative meaning. Cases in natural languages encompass meanings such as "by what means?" (instrumental; for example, in Sanskrit), "at what place?" (locative; for example, in Finnish), and "from where" (ablative; for example, in Latin)

A case can also straddle this boundary. An example of this would be a dative case (such as the one in Latin. This case can generally be translated with the English word "to", which keeps the same semantic/grammatical ambiguity.), which can both function as an indirect object marker (grammatical) or as the beneficiary of an action (semantic).

Example: "he gave the bone *to* the dog"

	lupus ("wolf")		fēmina ("woman")		victor ("victor")	
	Sing	Plur	Sing	Plur	Sing	Plur
Nom	lupus	lupī	fēmina	fēminae	victor	victōrēs
Acc	lupum	lupī	fēminam	fēminās	victōrem	victōrēs
Gen	lupī	lupōrum	fēminae	fēmināru m	victōris	victōrum
Dat	lupō	lupīs	fēminae	fēminīs	victōrī	victōribus
Abl	lupō	lupīs	fēminā	fēminīs	victōre	victōribus

Table 3.1: Case in Latin nouns (after Weiss, 2009, ch. 21-26)

3.3 Case and word order

If a language exhibits case, then it can have far greater freedom in word order in sentences. As shown above, in English, word order is used to convey a lot of information about the structure of a phrase. If, in another language, that information is conveyed explicitly through case, then the word order of the sentence is free to be used to carry other meanings.

One language that uses comparatively free word order is classical Latin. This section presents first a brief overview of the case system of Latin, and then two examples that show two ways that free word order can be useful. The first use discussed is about allowing an author's meaning to be clearer when they are writing prose. It is demonstrated by a brief examination of Caesar's writing about his conquests in Gaul. The second use is that in poetry, where there is already a conventional form defining how the lines of poetry should sound, and the words are being inserted into that framework.

3.3.1 The Case System of Latin

To understand any Latin writing is it necessary to have at least a cursory understanding of the Latin case system. Therefore, this case system is summarised here to render the subsequent discussions of Caesar and Virgil comprehensible. The definitions presented here apply only to Latin: other case systems will be discussed in summary later in the chapter.

Latin nouns and adjectives exhibit five cases[1]. The forms of some nouns are summarised in table 3.1.

The nominative, abbreviated as Nom, is the form of the noun one would find in a dictionary. The nominative marks the subject of a transitive verb:

[1]There is also a sixth form, called the *Vocative*, that is only used when directly addressing someone. This form is traditionally counted among the cases of nouns. However, the wider family of noun forms it fits into have a very unclear relationship with other cases (Spencer and Otoguro, 2005; Daniel and Spencer, 2009). Therefore, it is ignored here.

puella	canem	habet
*girl.*NOM	dog.ACC	has
a/the girl has a/the dog		

It also marks the single argument of an intransitive verb.

servi	currunt
*slaves.*NOM	run
the slaves run / the slaves are running	

In grammars, these two roles are usually clumped together under the name of the "subject".

THE ACCUSATIVE, abbreviated as ACC, primarily marks the object of a transitive verb.

puella	*canem*	habet
girl.NOM	*dog.*ACC	has
a/the girl has a/the dog		

The accusative is also used for the destination of an action of movement.

Romam	eo
*Rome.*ACC	I.go
I go to Rome	

It can also be used to denote extent in time or space.

arbor	erat	*septem*	*pedes*	altus
tree.NOM	was	*seven*	*feet.*ACC	high.NOM
the tree was seven feet high				

The accusative is also required after some prepositions, such as *ad*, "to" or "towards"; *apud*, "among" or "in the house of"; *ob*, "because of"; and *trans*, "across".

THE GENITIVE, abbreviated as GEN, is primarily used to mark possession or categorisation of nouns.

canis	*puellae*
dog.NOM	*girl.*GEN
the girl's dog	

turba	*luporum*
crowd.NOM	*wolves.GEN*
a crowd of wolves	

It is also used to indicate the whole of which another noun is a part (compare the English "a slice of pie").

plus	*vini*
more	*wine.GEN*
more wine	

The genitive is also often used to mark the location at which something took place.

Romae
Rome.GEN
at Rome

There are also some verbs and adjectives which must be accompanied by a noun in the genitive, such as *pudere*, "to be ashamed of"; *accusare*, "to accuse"; *memor*, "mindful of"; and *similis*, "similar to".

The DATIVE, abbreviated as DAT, is used to mark the indirect object. This can mark, for example, a transfer to a person or away from a person.

donum	*uxori*	dat
gift.ACC	*wife.DAT*	he.gives
he gives a gift to his wife		

pecuniam	*mihi*	ademit
money.ACC	*me.DAT*	he.took
he took money from me		

As a generalisation of this idea of transfer, the object of feelings or thoughts is often in the dative.

puellae	credo
girl.DAT	I.believe
I believe the girl.	

The datives of some nouns that name feelings can be translated as "an object of...".

odio	esse
*hatred.*DAT	to.be
to be an object of hatred	

In addition, there are a number of verbs which require a noun in the dative case, such as *imperare*, "to give orders to"; *studere*, "to be devoted to"; *placere*, "to be a pleasure to"; and "nocere", to harm.

THE ABLATIVE, abbreviated as ABL, has a bewildering variety of different uses. Generally, however, it carries one of three meanings. The first is "movement away from".

ex	*horto*	ambulat
out.of	*garden.*ABL	he.walks
he walks from the garden		

It is also used for the name of the instrument or tool that is used to perform the action denoted by the verb:

hostem	*gladio*	necavit
enemy.ACC	*sword.*ABL	he.killed
he killed the enemy with a sword		

This extends to techniques or skills, where in English it would usually be translated with the word 'by':

hostes	*virtute*	vincamus
enemy.ACC	*courage.*ABL	we.shall.beat
We will beat the enemy by courage!		

In other cases, the noun in the ablative is preceded by a preposition, and it is this that specifies the meaning.

In summary then, the case system of Latin divides the functions that nouns can perform in sentences into five categories, and identifies the category that any given noun in the sentence is inhabiting by a modification to the noun, generally by adding a suffix to the noun. This category is not part of the meaning of the noun

itself: it is a kind of metadata indicating the function that the noun is performing in the specific sentence of which it is a part.

Because this metadata is made explicit in the forms of the nouns, word order can be much freer, which can help the speaker of the language to use word order for other purposes, such as clarity, sound effects, discourse structure, or the demands of a predetermined form. Two examples of these from the Latin literature are described immediately below.

3.3.2 Prose: Caesar's *de Bello Gallico*

Julius Caesar's *Commentarii de Bello Gallico* ("Commentaries on the Gallic Wars") is an account of the military expeditions that Caesar led against the Celtic peoples of Gaul. It was published in Rome some time before 46 BC (Radin, 1918).

The *de Bello Gallico* is in very clear, unornamented prose. It was written to communicate, as clearly as possible, what Caesar wanted people to think had happened in Gaul. Because of the clarity of its prose, the *de Bello Gallico* has often been used in the teaching of Latin.

Caesar used word order for clarity in two distinct ways (Walker, 1918). In general, his sentences begin with the word that links them to the previous sentence. Because of the freedom of word order, this word can be a noun, or an adjective, or a verb. Sometimes, however, for emphasis, a different word is placed at the front of the sentence.

The two examples below belong to the first category. The Latin text is taken from Walpole's (1882) edition. Examples are all taken from chapter 2.

(2) Apud Helvetios longe nobilissimus fuit et ditissimus
Among Helvetii.ACC by far most-noble.NOM was and most-wealthy.NOM
Orgetorix. *Is* M. Messala et M. Pisoni consulibus...
Orgetorix.NOM. *He.NOM* M. Messala.ABL and M. Piso.ABL consuls.ABL...
Among the Helvetii, Orgetorix was by far the richest and most noble. When Marcus Messala and Marcus Piso were consuls, *he*...

In example 2, the first word of the second sentence is *is* ("he"), which creates a link back to Orgetorix, the scheming nobleman introduced in the first sentence. Orgetorix is also the grammatical subject of the second sentence. Grammars of Latin tend to assert that, in general, the subject comes first in Latin sentences (for example, see Weiss, 2009, ch. 40), so it is not surprising to see this here. However, in Latin, the pronoun can be omitted; so the explicit insertion of the pronoun and its positioning here does emphasise the link to the previous sentence.

Caesar does not limit himself to putting the subject first; if another word is the link to the previous sentence, he uses that instead.

(3) ... perfacile esse, cum virtute omnibus praestatarent, totius
... very-easy to-be, as in-virtue all.DAT they-excelled, all.GEN
Galliae imperio potiri. *Id* hoc facilius
Gaul.GEN empire.ABL gain-possession-of. *It.ACC* this.ABL more-easily
eis persuasit ...
them.DAT he-persuaded ...

... as they exceeded everyone in valour, it would be very easy to gain possession of an empire of all Gaul. By these means, he persuaded them more easily of this ...

In example 3, the grammatical object is first, followed by the instrument by which this is done. Both of these are links back to the previous sentence. There is no explicit grammatical subject, which is not unusual in Latin; instead, the verb indicates the subject.

In other places in the text, Caesar places other cases, and even verbs in sentence-initial position if they are the link to the previous sentence. This is a strong departure from the "basic" sentence structure.

Example 4 is an example of the second way in which Caesar uses word order (Trowbridge, 1907). This sentence is the first to mention Caesar, who famously speaks of himself in the third person throughout the book.

(4) *Caesari* cum id nuntiatum esset, eos per provinciam
Caesar.DAT when it.NOM announced were, they.ACC through province.ACC
nostram iter facere conari ...
ours.ACC to-march try ...

When it had been announced to Caesar that they were trying to march through our province...

Caesar puts his own name at the front of the sentence. By doing this he marks his own name as the most important thing in the sentence, even though it is neither the subject nor the object of either of the first two verbs, and nor is it a link to any previous sentence.

3.3.3 Poetry: Virgil's *Aeneid*

Virgil's *Aeneid* is an epic poem which was written during the reign of Augustus. It is largely a propaganda piece, concerning the mythical origins of the Roman state (Green, 2004).

Classical Latin poetry is based on a very different set of assumptions from modern English poetry. It forms its patterns not by the stress of the words or by rhyme, but by using long and short syllables.

The words "long" and "short" here do not have a specialist meaning. They refer to the length of time that a syllable takes to say. Examples of both long and short syllables can be found in the English name "Walker". The first syllable takes a lot longer to say than the second, because the 'a' sound is longer.

The rules for how lines of poetry can be put together from long and short syllables are complicated, and a summary here would not be useful (for such a summary, see any competent Latin textbook; the quick reference the author uses is at the beginning of Kennedy and Davis, 1964). For the purposes of this thesis it is enough to say that a line must obey one of a fixed number of patterns of short and long syllables.

By choosing the pattern for each line carefully, the poet could make each line have a sound appropriate to the thing it is describing. For example, the Aeneid is a poem mostly about warfare. Lines in the Aeneid that deal with melancholy concepts tend to be composed of long vowel sounds (consider the sound of the colloquial English phrase "doom and gloom", which uses a similar sound effect). Lines that talk about the galloping of horses, for example, can use staccato patterns of short and long syllables to make the sound of the words echo the actual sound of horses' hooves; lines about moving water can use repeated soft consonants to evoke the sound of waves lapping (Whately, 1945).

Fulfilling the requirements of the poetic style would be hard enough if word order were fixed; generating the desired sound effects would be almost impossible. Therefore, poetry tends to depart significantly from the "default" word order.

(5)

saxa	vocant	Itali	mediis	quae	in	fluctibus
rocks.ACC	call	Italians.NOM	middle.ABL	which.ACC	in	waves.ABL
aras						
altars.ACC						

Rocks in the middle of the waves that the Italians call "The Altars"

In example 5 (analysis from Clackson, 2008), Virgil has used a very atypical word order. The *quae*, meaning "which", which goes with *saxa* ("rocks"), would normally be next to *saxa*. Instead, it is a long way to the right, in the middle of a phrase (*mediis ... in fluctibus*) meaning "in the middle of the waves". Even if the *quae* is removed, this phrase is in an atypical order. Generally, *in* would precede the noun phrase it is attached to; here, it is in the middle of it.

3.4 Languages surveyed

The subset of mobile developers who know Latin well may well not be a large one (although the author is unaware of any research on this area). If case were

restricted to Latin, then its application outside that language would be dubious, let alone its application outside of language as a whole, as case would be reduced to an arbitrary outgrowth of linguistic history that just happened to appear in Latin. There would be no reason to suppose that developers would be able to use it in any intuitive fashion. It would be far more likely for developers to be able to use case intuitively if it were a widespread phenomenon among languages in the world: it is then more likely either that case can operate as a proxy for some deeper underlying way that people think about the interrelations between objects or that developers will have come into contact with the concepts involved through contact between languages and cultures. Whichever of these two is true, they will be more likely to be able to use case.

Fortunately, the use of case in the languages of the world is not restricted either to Latin or even to the language family of which it is a part, and the case systems of the languages of the world show some striking patterns in the kinds of cases they contain and what they are used for. These widespread patterns can then for a base for the use of case in plastic interfaces for the reasons suggested above.

To draw out these common uses for cases requires a survey of languages. There are very many languages in the world; at the time of writing, the current edition of the SIL Ethnologue (Lewis et al., 2013) contains information on 7,105 living languages, and there are certainly many languages that exist that have never been documented at all.

A large subset of these languages exhibit case. Blake (1977) surveyed 116 languages which exhibited case among the Australian languages alone. Therefore, the twenty languages discussed here should be taken as an illustrative, rather than as an exhaustive, survey. They have been chosen to provide evidence from a variety of times, places, and families of languages. Further information on these languages, arguing that they do not all receive their case systems from one another, is in appendix B.

Awngi is a language spoken in the north-west of Ethiopia (Lewis et al., 2013) with around 489,000 speakers as of the census of 2007 (Central Statistical Agency - Ethiopia, 2007). There are not many languages in Africa which have a complex system of cases (König, 2009); but König (2008) noted Awngi as a language with an unusually expressive set of cases by African standards.

Azoyú Tlapanec is a language spoken in the west of Mexico. Lewis et al. (2013) quote the Mexican *Instituto Nacional de Lenguas Indigenas* as saying that the language has around 590 speakers; but they also note that this number is likely to be inaccurate because many people who speak this language are unwilling to admit that they do.

Basque is a language spoken on the France–Spain border and in the north of Spain. It is estimated to have around 500,000 speakers in this area, all bilingual (Trask, 1997; de Rijk, 2008).

Etruscan is an extinct language which was spoken until around 50 AD in the north of Italy, eventually being displaced by Latin. It is known from around 12,000 inscriptions and six longer texts (Wallace, 2008, ch. 1)

Finnish is one of the two national languages of Finland, as laid down in the Finnish constitution (1999, ch. 2, s. 17). It has nearly five million speakers (Lewis et al., 2013).

Georgian is the national language of Georgia, and is also spoken in Iran and Turkey. It has around four and a quarter million speakers (Lewis et al., 2013).

Greek is a language with 3,300 years of attested history. Remarkably, through all this time, it has remained recognisable (Fortson, 2009, para 12.1). Its modern form is the national language of Greece, and is spoken by around 13 million people (Lewis et al., 2013). The discussion below mentions two varieties of Greek: *Koine Greek*, the first dialect of Greek to be used across the whole Greek world, and the language of the New Testament (Woodard, 2008; Wenham, 1965); and *Dhimotiki*, the current official version of the Greek language (Adams, 1987; Sofroniou, 1962)

Hungarian is the national language of Hungary, spoken by around 12 million people (Lewis et al., 2013).

Icelandic is the national language of Iceland, spoken by around 240,000 people (Lewis et al., 2013).

Kalkatungu is a recently extinct language that was spoken in Queensland, Australia. Blake's grammar of Kalkatungu (1979) was based on the speech of the last dozen speakers; by the time this grammar was published, only one fluent speaker remained alive, and by 1983 there were no known fluent speakers remaining. Blake recounts:

> *The elderly speakers we consulted were all very willing to be recorded. They were all familiar with recording inasmuch as some of their friends and family owned recorders and they seemed to see some value in having their language recorded knowing that they were the last speakers. (Blake, 1979, p. 3)*

Khalkha Mongolian is the official language of Mongolia. It has around 2,373,000 speakers (Lewis et al., 2013).

Latin was the language of Rome and the Roman Republic and Empire. It is an extinct language: there are now no people whose first language is Latin. However, a small number of people speak it fluently as a second language (Lyman, 2005); it is important in the Roman Catholic church, who publish a dictionary of Latin neologisms for modern concepts (Opus Fundatum "Latinitas", 1992); it is still actively taught in schools; and modern literature is still occasionally made available in the language (Milne and Lenard, 1960).

Modern Standard Arabic is the official form of the Arabic language, used as the official language in 20 countries. It is the only standardised version of a large spectrum of dialects. The various dialects of Arabic are spoken today by around 223-250 million people (lower figure from Lewis et al., 2013, higher from Holes, 2004).

Northern Sami is a language largely spoken in the north of Norway, but also spoken in the north of Sweden and Finland. It is spoken by around 20,700 speakers (Lewis et al., 2013).

Old English was the form of English spoken by the Anglo-Saxons from their arrival in England until around the time of the Norman invasion (Quirk and Wrenn, 1955).

Russian is the national language of the Russian Federation. It is spoken by around 160 million people (Lewis et al., 2013).

Sumerian is the oldest language attested. It has the longest literary tradition of any ancient language, lasting around 3000 years from around 3200 BC (Michalowsky, 2008). The modern understanding of Sumerian is not complete; nonetheless, two descriptive grammars have been produced recently (Edzard, 2003; Jagersma, 2010).

Tocharian A and B are two closely-related extinct languages which were spoken in the north of China and are attested from the 6th century to the 8th century AD (Adams, 1988, ch. 1). The speakers of Tocharian left nearly nothing written down about themselves, and so the question of who actually spoke this language is an open one. The texts in Tocharian that have survived are largely Buddhist religious texts (Fortson, 2009, s. 17.3).

Tsez is a language spoken in the Dagestan region of Russia and in Georgia, with around 12,500 speakers in Russia (Lewis et al., 2013).

Turkish is the official language of the Republic of Turkey, defined as such in its constitution (1995, article 3). It is spoken by nearly 51 million people (Lewis et al., 2013).

3.5 Kinds of meanings expressed through case

Every language is different and, of the languages that have case, every language's case system is different. However, there are some startling similarities in case systems that are apparently unrelated. Specifically, there are some very common meanings and functions that commonly appear in case systems cross-linguistically.

Again, the survey in this section is not exhaustive. Instead, these are the patterns that introduce the concepts necessary for the rest of this chapter and the main body of this thesis.

3.5.1 Core syntactic verb arguments

The syntactic arguments of a verb are the noun phrases that appear with it that are grammatically necessary. The core syntactic arguments of a verb are its subject and, if it is a transitive verb, its object.

Example: "*The dog bit" is missing an object argument, and is thus ungrammatical.

Example: "*Bit the boy" is missing a subject argument, and is thus ungrammatical.

Languages which use case have a tendency to use cases to mark the subject (referred to as A, for "agent") and object (referred to as "P", for "patient") of transitive verbs, and the single argument of intransitive verbs (referred to as "S", for "single").

There are three common patterns for how this is arranged. Every one of the languages in this chapter exhibits one of these three patterns. The first and most common (Blake, 2001, ch. 5) is called "nominative-accusative", and refers to languages where the subject of a transitive verb is in the same case as the single argument of an intransitive verb. This case is traditionally referred to as the "nominative". The object of a transitive verb is then in a second case, traditionally referred to as the "accusative".

This pattern can be seen clearly in English pronouns:

Example: "*He* approached her" and "*He* slept". "He" is the nominative form, used for S and A.

Example: "She approached *him*", but never "**Him* slept". "Him" is the accusative form, used for P only.

The languages of the twenty which use the nominative-accusative pattern are: Awngi (Hetzron, 1978), Etruscan (Wallace, 2008, ch. 5), Finnish (Karlsson, 1999, chs. 7 and 9), Greek (Adams, 1987, p.15; Wenham, 1965, p. 9), Hungarian (Rounds,

2001, sec. 6.1), Icelandic (Einarsson, 2000, pp. 105–107), Khalkha Mongolian (Svantesson, 2003), Latin (Wilson, 1968, p. 62), Modern Standard Arabic (Holes, 2004, ch. 4), Northern Sami (Sveriges Utbildningsradio AB, 2011, pp. 9–10), Old English (Quirk and Wrenn, 1955, pp. 59–61;Marsden, 2004, p. 377), Russian (Wade, 1992, pp. 85–86), Tocharian (Adams, 1988, ch. 5) and Turkish (Lewis, 1985, pp. 35–36).

The second pattern is called "ergative-absolutive", and refers to languages where the *object* of a transitive verb is in the same case as the single argument of an intransitive verb. This case is traditionally referred to as the "absolutive". The *subject* of a transitive verb is in a case of its own. This case is traditionally referred to as the "ergative". The Basque language exhibits this pattern (de Rijk, 2008, chs. 2 and 9). The absolutive case has no ending:

Peruk	*ogia*	dakar
Peter.ERG	*the bread*.ABS	is bringing it
Peter is bringing *the bread* (de Rijk, p. 198, simplified)		

Gizona	dator
The man.ABS	is coming
The man is coming. (de Rijk, p.198)	

The ergative case can be distinguished by its -k ending:

Peruk	ogia	dakar
Peter.ERG	the bread.ABS	is bringing it
Peter is bringing the bread.		

Other languages which exhibit this pattern include Azoyú Tlapanec (Wichmann, 2004), Tsez (Comrie et al., 1998, pp. 18–19) and Sumerian (Edzard, 2003, pp. 35–36; Jagersma, 2010, pp. 154–160 and 295–296).

The third pattern is "split ergativity". Languages which exhibit this pattern are those in which some words behave in an ergative-absolutive way and some behave in another way, often a nominative-accusative way. Two languages which exhibit this pattern are Kalkatungu, which has ergative-absolutive-like nouns but some nominative-accusative-like pronouns (Blake, 1979, pp. 36–37 and 41–42) and Georgian, which determines whether a noun or pronoun should exhibit nominative-accusative-like behaviour or ergative-absolutive-like behaviour based on the properties of the nearest verb to which it is attached (Aronson, 1990, secs. 2.3, 3.1.1.2 and 5.1).

3.5.2 The indirect object and the beneficiary

The indirect object of a verb is another noun that the verb is acting on besides the subject and the object . This usually implies an idea of a transfer of some description. In English, it is marked with the preposition "to":

Example: She gave the ball *to the dog*

Example: He spoke *to the greengrocer*

Languages often use a case to mark the indirect object of a verb. These cases are traditionally referred to as "datives". Cases that encode the grammatical indirect object are found in Turkish (Lewis, 1985, p. 36), Kalkatungu (Blake, 1979, pp. 44–45), Khalkha (Svantesson, 2003, p. 163), Georgian (Aronson, 1990, para. 7.2.3) and Tsez (Comrie et al., 1998, p. 19)

This case often encodes the beneficiary or the target of a transfer even when it is not grammatically an indirect object. This can be found in Latin (Scottish Classics Group, 1996, p. 10), Koine Greek (Wenham, 1965, p. 245), Old English (Marsden, 2004, p. 377), Icelandic (Einarsson, 2000, p. 109), Basque (de Rijk, 2008, p. 346), Russian (Wade, 1992, pp. 100–103), Sumerian (Edzard, 2003, p. 40) and Hungarian (Rounds, 2001, pp. 112-113).

The case of Etruscan is unclear. It has a case which seems to encode an indirect object, which Bonfante and Bonfante (2002) tentatively named as a dative; however, both (Rix, 2004) and Wallace (2008) refer to it as a "pertinentive" and argue that its functions are not sufficiently well known to make any strong statements.

3.5.3 Possession and category

Languages often use a case to mark the possessor of an object. English retains this on pronouns:

Example: *Her* cat

These cases are traditionally referred to as "genitives". Cases specifically for possessors can be found in Basque (de Rijk, 2008, p. 100), Modern Standard Arabic (Holes, 2004, p. 172), Northern Sami (Sveriges Utbildningsradio AB, 2011), Tsez (Comrie et al., 1998, p. 6) and Turkish, the with the additional restriction that the possessor must be a concrete person or object (Lewis, 1985, pp. 36 and 41–42).

Many languages generalise this pattern and use this case to express meanings about categorisation, such as:

- The name of a category or a set to which an object belongs.

- The name of an attribute of an object or of an abstract idea that an object exemplifies.
- The name of a context in which an object must be understood.

Patterns of this kind can be found in Finnish (Karlsson, 1999, pp. 95–97), Georgian (Aronson, 1990, sec. 3.3.1), Koine Greek(Wenham, 1965, p. 245), Icelandic (Einarsson, 2000, p. 111), Khalkha (Svantesson, 2003, p. 163), Old English (Quirk and Wrenn, 1955, pp. 61–62) and Russian (Wade, 1992, pp. 87–92).

Latin and Etruscan have similar patterns, but in a more restricted way: Latin uses its genitive case to indicate that a person has a quality (compare the English "a creature of habit"; Wilson, 1968, p. 63), and Etruscan uses its to mark the name of the family to which a person belongs (Wallace, 2008, pp. 96–97).

In Sumerian, as well as marking the possessor, the genitive marks a noun part of which is being acted upon by the verb, comparable to the slightly archaic English "she gave me of the tree, and I did eat" (Gen 3:12, KJV).

These patterns all represent a relationship between two objects where the noun in the genitive represents a whole, and its head represents a part of that whole.

Some languages generalise the genitive still further, to encapsulate any asymmetric relationship between two nouns. This can be found in Awngi (Hetzron, 1978, p. 126).

3.6 Blake's case hierarchy

Based on an extensive survey of case systems in languages, (Blake, 2001, ch. 5) identified around 40 basic "case roles" which are instantiated in languages worldwide. A "case role" is a basic role that a case can perform. He then produced what he referred to as a "case hierarchy". The case hierarchy is a model which encompasses the kinds of case system found in the languages he surveyed, and has predictive power so that it can be falsified against languages discovered in future. The case hierarchy has withstood examination and further data well (Malchukov and Spencer, 2009).

The case hierarchy is an ordered list of the case roles that cases tend to have (given in table 3.2) arranged so that if a language has a case fulfilling a role that is on the list, it will usually have all cases that encompass all the roles above that case on the list. The lowest case on the list that the language has is likely to have a range of functions with no unifying principle.

So, for example, because English has a distinguishable genitive (see section 3.5.3), it is very likely to have distinguishable nominative and accusative or ergative

Nominative
Accusative or Ergative
Genitive
Dative
Locative
Ablative or Instrumental
Other cases

Table 3.2: Blake (2001)'s case hierarchy

Nominative
Accusative or Ergative
Genitive
Dative
Locative
Ablative or Instrumental
Other cases

Table 3.3: The case hierarchy applied to Icelandic

cases as well, because they are above genitive on the hierarchy. This matches what can be found in English pronouns.

Icelandic, by contrast, has a nominative-accusative system of four cases. By applying Blake's case hierarchy, one can make an educated guess that it will probably have a nominative, an accusative, a genitive and a dative, and that the dative will have a variety of extra functions (figure 3.3). This is indeed the structure of the case system in Icelandic (Einarsson, 2000).

Likewise, if one starts at the top of this list and picks any number of cases, the outlines of a case system will emerge which appears in at least one natural language.

3.7 Why case may be applied to plastic interfaces

So far, this chapter has argued that case exists, that it is not restricted to one time, place or language family, that it shows patterns of common meanings, that these patterns are predictable and that these patterns can be grounded in linguistic theory. What it has not yet argued is its applicability to plastic user interfaces. This section discusses this problem.

Case, being a linguistic structure, might purely be applicable to language *sensu strictu*. If it were purely to do with the inner workings and the machinery of language, then it would likely not be applicable outside of language proper. However, two situations were alluded to above where it could be useful outside of language:

Position	Language	Speakers (millions)
1	Chinese	1,197
2	Spanish	406
3	English	335
4	Hindi	260
5	Arabic	223
6	Portuguese	202
7	Bengali	193
8	Russian	162
9	Japanese	122
10	Javanese	84.3

Table 3.4: 10 most spoken languages in the world (Lewis et al., 2013)

either where developers use it analogously from their existing familiarity with the kinds of concepts that it talks about, or where the case systems of languages act as imprecise proxies for an underlying set of semantic categories that people use to think about objects, actions and their interrelations.

3.7.1 As an analogy

The first reason is that case may be useful analogously. If languages with case systems that correspond to Blake's case hierarchy are suitably widespread, then the categories they contain will already be familiar to developers who speak those languages.

Table 3.4 reproduces the table of the ten most populous languages in the world from the 17th edition of Ethnologue (Lewis et al., 2013). Of these, only Russian (Wade, 1992) and Japanese (Ogawa, 2009) have complex case systems today; but the concept of case is entirely alien only to Chinese and Javanese (Lee and Thompson, 1987; Peyraube, 2004; Suharno, 1982). Arabic has a system of three cases (nominative, accusative and genitive) which have been consistently weak since the classical period (Holes, 2004). The remaining five languages—Spanish, English, Hindi, Portuguese and Bengali—are all Indo-European languages which have lost most of their case-marking in historical times due to phonological change (Cardona, 1987; Blake, 2001, s. 6.3).

Case may still be useful as an analogy, however, in those languages. In Indo-European languages which are losing case, the meanings that were attached to cases often get attached to prepositions. One often sees descriptions of cases which map them directly to English prepositions, especially in teaching grammars. Genitive cases are often introduced with wording such as "If you see a word in the genitive case, you should preface it with the word 'of'." (Jones, 1998, p. 88) and "The Icelandic genitive corresponds both to the English *-s* genitive and the

of genitive" (p. 110 Einarsson, 2000, emphasis in original). Likewise, dative cases are often introduced with wording such as "The Dative [sic] is ... often shown in English by *to* and *for*." (Wilson, 1968, p. 64, emphasis in original).

3.7.2 As clues to underlying universal semantic roles

The second reason is that cases may actually encode underlying semantic roles that are not limited to use in language. The theories summarised above are the major ones that argue this. None of the theories that argue this, however, agree on which semantic roles are the universal ones, on how these relate to other elements of human thought, nor even on which cases should be considered to have semantic content. However, in each theory, the semantic roles have been extracted largely from the structure of the case systems of natural languages.

If language is not separate from other processes by which people communicate, there is no reason why the proposed universal semantic roles must be confined to language rather than being present in other symbol-systems that people create. More specifically, they may be applicable in user interface design as well (for theoretical approaches to interface design based on a semiotic or sign-based approach see Goguen, 1999; de Souza, 2005).

In this situation, case may be used in the sphere of user interfaces as a proxy for an underlying set of universal semantic roles.

There are a number of mainstream theories that deal with case as proxies for underlying sets of semantic roles. These are generally not purely theories about case. They are trying to explain a set of phenomena in which case plays a role. These theories hold that there is a set of underlying primitive semantic structures that surface through case (though not exclusively through case in all languages), and that these structures are reflective of human cognition as a whole, not just language; others hold that these roles are valid cross-linguistically and make no statement about whether they are applicable outside of language itself. Two major semantic theories that embrace case as symptomatic of an underlying set of semantic roles are outlined below.

It is worth noting that these theories do not have the same predictive power over the structures of case systems as Blake's case hierarchy. However, this is not their purpose: their domain is larger than just the structures of case systems.

3.7.2.1 Cognitive grammar

Cognitive grammar is a theory that holds that language is integrated into human cognition, rather than is separated off into a specialist area of the brain. Because of this, everything in language has a meaning, rather than grammar just being a framework on which meaning-bearing words are hung. Grammar as a whole is

attached to the human ability to think in symbols; grammatical structures can have meaning in exactly the same ways that words do (Langacker, 1986). It holds that case systems are based on a cross-linguistic universal set of semantic roles (Blake, 2001, p. 62).

In Cognitive Grammar a case sets up a "trajector-landmark asymmetry". In a trajector-landmark asymmetry, two objects have an asymmetrical relationship where one of the objects (the "trajector") is put into the foreground using the other (the "landmark") as a background, or context.

Luraghi (2009) gives an example of this using the Latin phrase *domus patris*, "father's home" 6.

(6) domus patris
home.NOM father.GEN

In this example, the trajector is the house. The house is in the foreground, and is semantically the most important element in the phrase. The father provides a background or a context to the house; the house is to be considered in the context of the father owning it.

All cases set up a trajector/landmark asymmetry. The different cases, however, set up different kinds of asymmetry; the genitive example above sets up an ownership relationship, for example. The meaning of the case defines what kind of relationship is created.

3.7.2.2 Localist case grammars

Localist case grammars are approaches to the problem of grammar that share two characteristics. First, they hold that cases are semantic and meaningful, and that they expose an underlying set of semantic roles. Second, they hold that all these semantic roles are based on spa tial reasoning, and that all cases are, fundamentally, about location in space and about movement. "Abstract" uses of these roles are metaphors for their spatial use (Anderson, 1987). The semantic roles are universal across language (Blake, 2001, p. 62).

Anderson (2009) illustrates this with the phrase "Betty taught Bill that song". Here, Betty is a source from which something moves; Bill is a destination at which the song ends up; and the song is the thing to which the movement is happening.

3.7.2.3 Fillmore's Case Grammar

Fillmore's (1968) Case Grammar is one of a large family of theories that is concerned with "deep structure". The deep structure of a sentence is a theoretical model of the structure of that sentence, often some kind of tree (although the precise nature of the structure varies depending on the theory). In these theories,

for any given language, there is a set of tree transformations that turn a sentence of the language into a deep structure tree, and vice versa. No language has ever been entirely described in this way.

The reason why deep structure is posited at all is that it allows similar sentences to have similar deep structures even when their surface syntactic structure is quite different. A good example of this can be found in passive sentences. The two sentences *"The dog bit John."* and *"John was bitten by the dog"* are, in surface structure, quite different: the subject and object in the second are reversed compared to the first. The deep structure of these sentences, however, would be the same or very similar.

Case Grammar is a theory about verb valency in this deep structure. The valency of a verb is the number and nature of the places it has for nouns. For example, the verb "to bite" takes both an agent (the creature who does the biting) and a patient (the creature who is bitten). These are distinct from the concepts of "subject" and "object" described above: in the two sentences above, the subject and object are swapped in the second sentence compared to the first, but the agent and patient are the same in both sentences.

In Case Grammar, the kinds of relationship a noun and a verb can have *in the deep structure of the sentence* are categorised. Elements of this categorisation are called "deep cases", and the set of deep cases is a universal across languages. Case on the surface is one prominent way of generating deep cases in the deep structure of the sentence: word order (as in English) is another.

3.7.2.4 Natural Semantic Metalanguage

The above theories all propose a set of semantic primitives that explain relationships between nouns and verbs and that are engaged in some way with the structures of case systems. Natural Semantic Metalanguage ("NSM") takes a different approach: it aims to find a small set of "semantic primes" that can be used to explain all meaning-bearing features in any language and that exist cross-linguistically (Goddard and Wierzbicka, 2002) . The set of semantic primes includes ideas such as "I", "you", "something", "one", "two", "to be somewhere" and "to have". The primes have emerged from a very large amount of empirical work suggesting that these primes not only exist in all languages but combine in the same way in all languages (Wierzbicka, 2009).

NSM has been used with success in analyses of the meanings of individual cases (Wierzbicka, 1980, 2009; Blake, 2001) but it does not attempt to explain the structure of case systems.

That case can be used either in analogy or as a proxy for underlying semantic roles shows that the use of case in the context of plastic user interfaces is not necessarily a theoretical *faux pas* from the perspective of linguistics.

In chapter 2, a gap was identified in the tools available to mobile developers. Existing tools fail them in one of two ways: the tools available from the research community target more platforms than developers need, require the use of methods that are alien to their professional practice, and do not take into account the guideline-defined requirements on them. The industrial tools provide no adaptation at the dialogue layer.

In section 3.3 two examples were given of how case can be used linguistically to put words into an order that fit the requirements of the form they are in. This is where the analogy between word order and dialogue structure of an application becomes important; if each object that the user selects in a dialogue structure is annotated with a case, and if rules from the guidelines can be rephrased in terms of cases, then similar benefits of flexible selection ordering might be obtained. This is how case can be used to attempt to overcome the issue with the industrial tools.

Case can be used to attempt to overcome the issue with the research tools by its comparative simplicity. Even remarkably complex case systems, such as those in Finnish and Hungarian, have fewer than 20 cases (Rounds, 2001; Karlsson, 1999), and Blake's case hierarchy notes only five as common. Latin has only five; Attic and Koine Greek only four: and these languages used it to considerable effect for making word order flexible. This means that even if the meanings of the cases were to be entirely arbitrary, the developer would have only four or five meanings to learn that could be used as part of a known and trusted development methodology, instead of having to learn an entire MDE system that would displace their existing tool chains. The situation is in fact better than this, because the meanings of the cases in use in the tool need not be arbitrary. Comparative linguistics has provided a set of meanings that occur cross-linguistically and which may well be familiar for one or both of the reasons outlined above. If either of these are indeed the case, then the developer will not even have to learn the meanings of the cases, merely apply the meanings they already know.

These statements about how the gap can be bridged, however, are vague and wanting in precision. An example will clarify how case can be used to build versions of an application on platforms with different ordering assumptions.

3.9 Plasticising the dialogue component: a worked example

Below is an outline of a conceptual framework showing one way that case can be applied to interface design. This is the conceptual framework that will be developed later in the thesis. It cannot claim to be the only possible way of applying case to user interface design.

The Arch model, as discussed in chapter 2, contains a component which is responsible for task structure and the order in which users are able to do things. By its definition, the dialogue component is about ordering, and about the linear sequence of actions that the user performs to perform the task that they are intending to perform.

A simple but dramatic example of different platforms having different ordering conventions can be found by examining two drawing programs with comparable feature sets that embody different assumptions. These two programs are Xfig (which is still being maintained) and MacDraw (which is not).

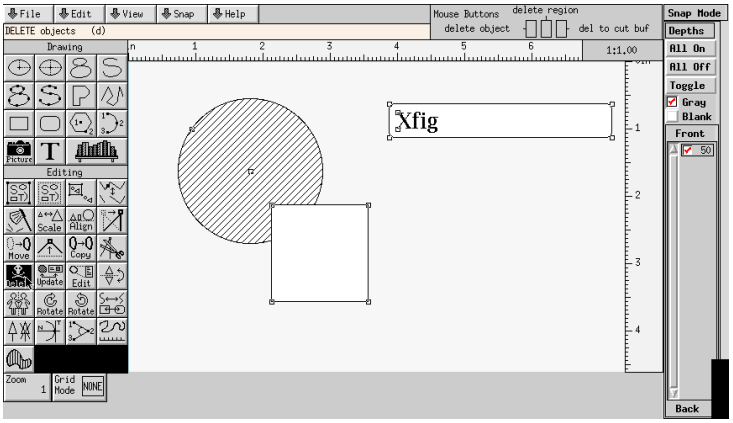
In Xfig, the ordering rule is simple. The user always chooses the tool that they want to use, and then selects the region of the drawing that that tool is to operate on. For example, to delete an object, the user chooses the delete tool and then either clicks on a single object to delete that, or drags out a region on the canvas to delete all of the objects in that region (figure 3.1).

In MacDraw, the ordering rule is slightly more complicated. Here, tools that create objects behave as they do in Xfig, putting the software in a mode in which the user can select an area of the canvas in which the new object should be created (figure 3.2). On the other hand, tools that adapt, change or delete an existing object require that object to be selected first, and only then can the tool (usually displayed as a menu item) be selected (figure 3.2).

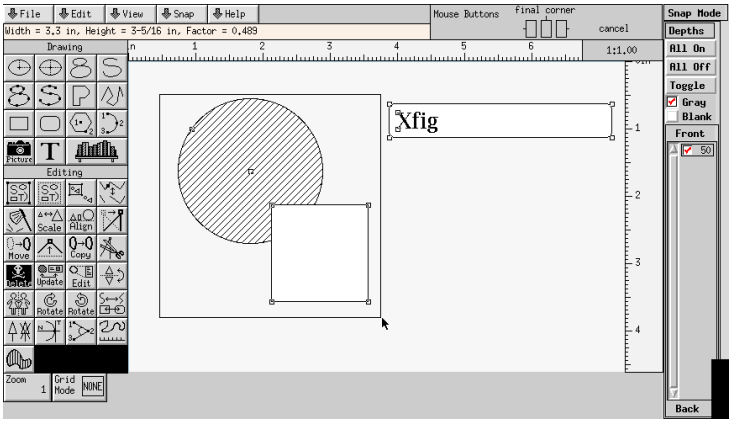
As shown above, in natural language, one of case's major uses is in allowing the words of a phrase to change order depending on the speaker's circumstances. To apply case to plastic user interfaces, these two situations—the one reordering task elements, and the other reordering words—are considered as comparable. This comparison is tenable so long as one of the two arguments in the previous section hold.

Case is applied to the individual objects that the user can interact with, according to the task structure. For example, a genitive case will be applied to an interaction which specifies an owner or a category; and a dative case will be applied to an interaction which specifies the destination of a transfer or a beneficiary. This would provide more information for the re-ordering process to work on than a simple, semantically un-annotated task structure (such as those in section 2.8).

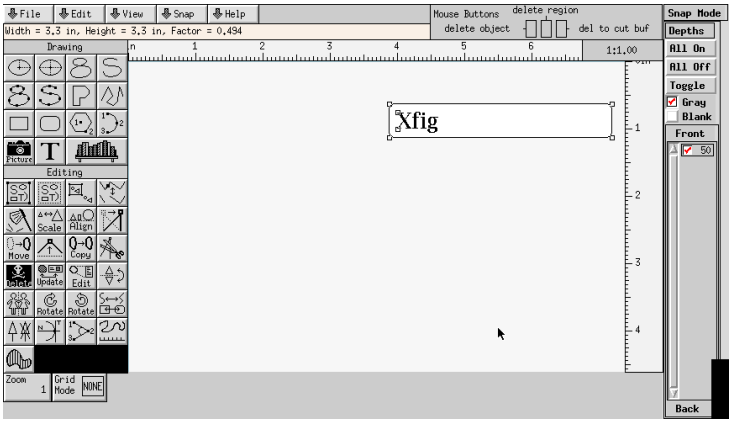
Applying this idea to the drawing programs yields the shadowy outline of a solution, assuming that some kind of machine-manipulatable declarative notation



(a) Select tool

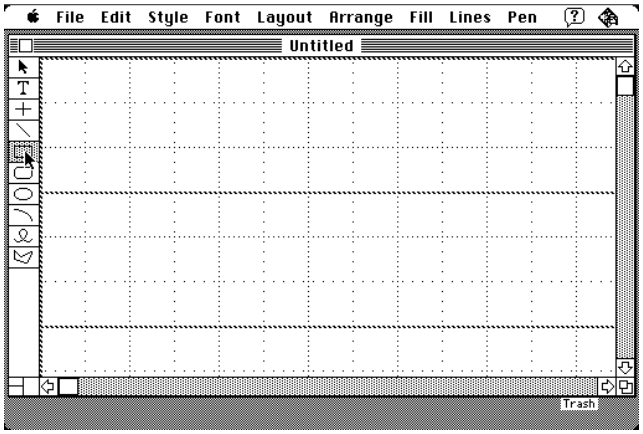


(b) Select region

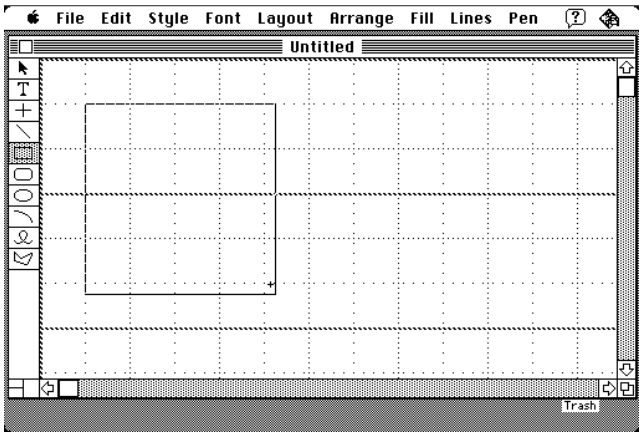


(c) Objects deleted

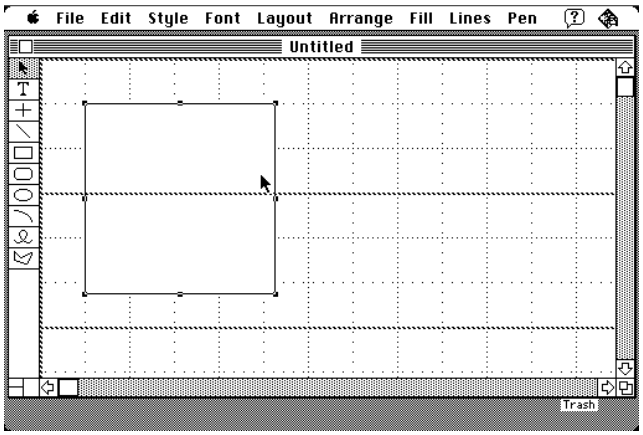
Figure 3.1: Object deletion in Xfig



(a) Select tool

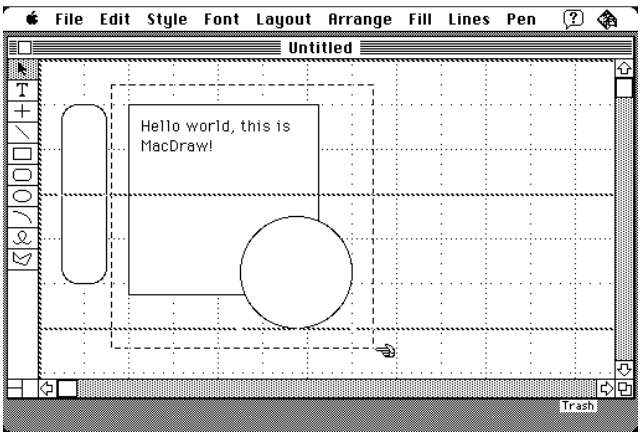


v(b) Select region

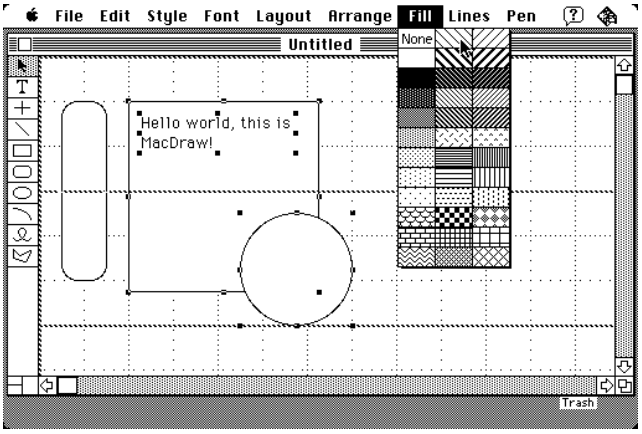


(c) Object created

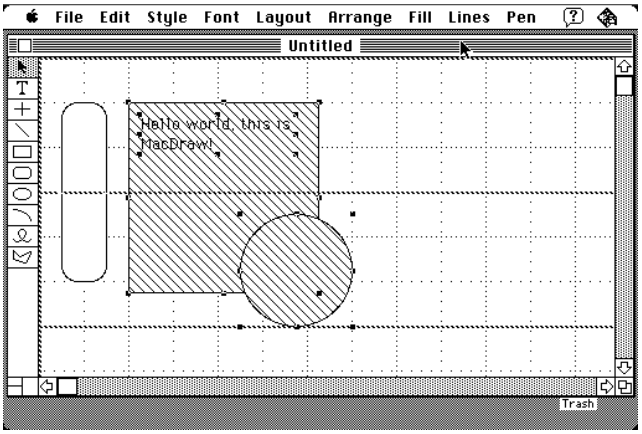
Figure 3.2: Object creation in MacDraw



(a) Select region



(b) Select tool



(c) Object modified

Figure 3.3: Object modification in MacDraw

is used for the dialogue controller. It is not possible to work back from the Xfig-style interface to the MacDraw style interface without knowledge of what each tool does in some detail; but it is possible to work from the MacDraw style of interface back to the Xfig style of interface without needing to know any details of the purposes of the tool. Hypothetically, one could assign an instrumental case to tool selection and a locative case to the selection of an area of canvas. A rule could then be applied to the dialogue controller that says if there are situations where the user would select a locative before the user selects an instrumental, then those two selections should be swapped, so that the user first selects an instrumental and after that a locative. Note that this rule is not application-specific: it instead encodes a broader statement about potentially many applications.

Some weaknesses are showing up in this mechanism even at this high level. It obviously does not deal well with dependencies: further annotations would be needed to mark situations where a selection actually must be in a specific place in the task structure. Also, one cannot run this in reverse without further annotations: there is no way to get back to the MacDraw interface from the Xfig interface, because information has been lost in the process of generating the Xfig interface. Even with these taken into account, however, the approach may provide a useful degree of plasticity.

This example, however, does not form any kind of rigorous argument for the usefulness of case, other than clarifying how it might be applied. Firstly, it does not talk about user interface *quality* at all, even with regards to a given set of user interface conventions. The rule applied to get from one interface to the other is application-specific and does not correspond to any rule laid down in the user interface guidelines for either platform. Secondly, it only outlines a single case, and that in isolation. Previously in this chapter, cases were shown to come in systems, and to demonstrate the usefulness of case requires the use of a case system that resembles a natural case system.

The case attached to an interaction in the task model can also be used by the logical and physical adaptation layers. On the physical layer of the Arch, the physical arrangement of elements on the screen may need to reflect the relationship between the data that they contain . A good example of this can be found in the iOS Human Interface Guidelines on the use of the "split view" screen design on the iPad, as shown in figure 3.4:

> *You can use a split view to display persistent information in the left pane and related details or subordinate information in the right pane. In this design pattern, when people select an item in the left pane, the right pane should display the information related to that item. ... Avoid creating a right pane that is narrower than the left pane. (Apple, Inc., 2012a)*

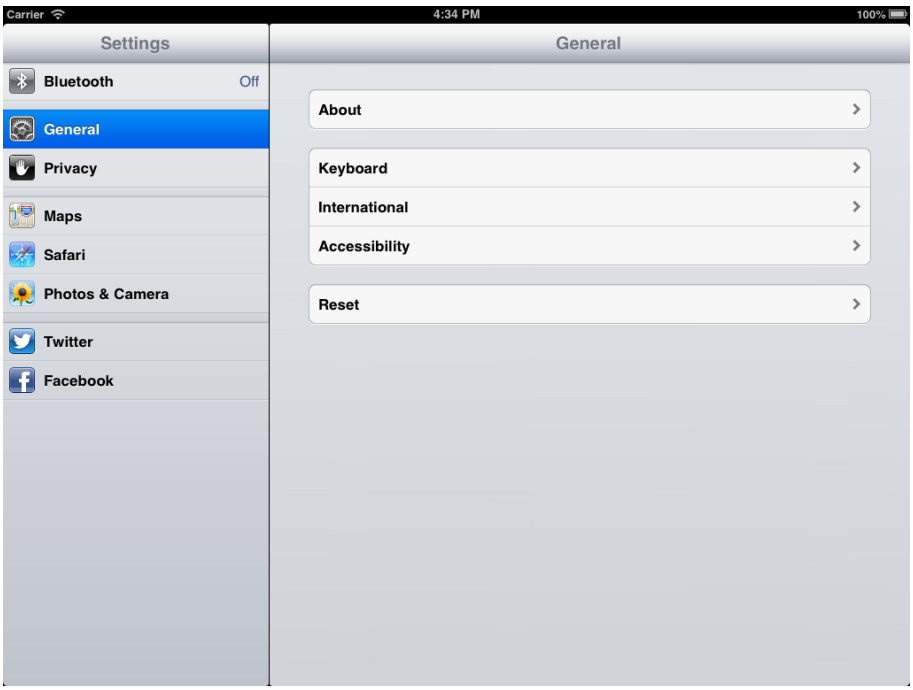


Figure 3.4: The split view on iOS

A genitive-like relationship certainly fits into the scope defined by this paragraph: the left pane allows the user to select the owner or the category, and the right contains the things within the application that are attached to that owner or category.

Its usefulness on the logical layer can also be deduced from the iOS Human Interface Guidelines. The operating system provides an "Activity View" which is used to display methods that an application can use to share content:

> *When users tap the Share button, a set of activities is presented by an activity view controller. ... An activity view controller displays a configurable list of services that the user can perform on the specified content. Users tap the Share button to reveal the contents of an activity view controller. ... On iPhone and iPod touch, an activity view controller appears in an action sheet; on iPad, it appears in a popover.*

This leaves the definition of an "activity" somewhat loose; but it can be clarified by considering the mention of the "Share" button, and looking at the developer documentation:

The system provides several standard services, such as copying items to the pasteboard, posting content to social media sites, sending items via email or SMS, and more.

The given activities are all, in some sense, destinations. The pasteboard is a destination in its own right, as are social media sites or printers. E-mail addresses and telephone numbers for SMS are also destinations. This pattern therefore covers dative patterns, and does so by specifying its representation in terms of the logical layer of the Arch model.

3.10 Research questions

At this point, enough information has been given above (both in this chapter and in chapter 2) to clarify the research questions and to specify in more detail what kinds of answers are to be expected to them.

Are the categories delineated by case relevant to user interfaces? In section 3.9 above, a sketch outline was given of application of categorisations based on case to talk about re-ordering of user interfaces. This sketch, however, is insufficient as an answer to this question for two reasons.

The first insufficiency of the sketch is that it takes an instrumental meaning (which is near the bottom of the case hierarchy) and uses it in isolation. However, in real languages, the meanings of cases do not exist in isolation. This chapter, and especially section 3.6 presents evidence that there is a *set* of meanings involved in case systems. These meanings are interrelated in ways that are very consistent across languages and contexts of use. This means that any system that purports to be based on case must take this interrelation into account.

Not all of the common meanings of case can be expected to be applicable to user interfaces. Section 3.5.1 laid out a set of meanings that are to do with verb valency—to do with attaching nouns to verbs with no regard for meanings. In the traditional analysis the nominative, accusative, ergative and absolutive cases do not *mean* anything at all. They are syntactic, in the linguistic sense of the word, and specific to language in the strict sense.

However, below these top two in the case hierarchy, semantic meanings are often found (see section 3.5 above). These categories are potentially applicable to the domain of user interfaces, as suggested in section 3.9. Therefore, to justify being called "case", the framework of categories that are being used to talk about user interfaces should follow the rules that cases follow as far

as is possible: specifically, they should adhere to the structure of the lower part of the case hierarchy, where semantic meanings exist.

The second insufficiency of the sketch is that it does not demonstrate that case is a living phenomenon. Cases, in natural languages, can be used in unexpected ways. For example, in section 3.9 above, the dative was described in terms of social media destinations or other electronic destinations that come under the definition of what Apple describes as Activities. However, if the dative can *only* be used for these things then it becomes, at best, just another way of talking about what Apple describes as Activity View Controllers. It is only when it is used by developers for unexpected things that it deserves to be considered as a separate phenomenon.

Thus, an answer to this question should demonstrate not only that the categorisation is useful for user interfaces but also that it deserves the name case: the categorisation in use must obey the patterns given above for case systems, and must be amenable to use in unexpected ways.

Can a tool be built that embodies case for building plastic user interfaces? A tool embodying case means that it can use the categorisations that case provides to do something. The cases must be implementable.

This implementability comes in two parts. The first part is that the tool itself must be able to deal with cases. It cannot simply be, for example, a text editor that recognises the names of cases and highlights them: the way that the case is to be used needs to be implemented as either an algorithm or as a rule in a grammar or as some similar executable representation. If the tool takes no actions based on cases, then those cases have not been implemented or embodied in the tool.

The second part is that the implemented cases should result in sensible user interfaces according to the user interface guidelines of the platforms involved. The importance of the user interface guidelines to mobile platforms was discussed in chapter 2. This is not a formal property of the tool. Instead, an answer to this question needs to be determined through the use of the tool: if the uses that developers come up with for the cases match the user interface guidelines, then case is implementable in terms of those guidelines.

Can case as embodied in this tool be used to build useful applications? The end point of a mobile developer's involvement in a project is usually the handing over of the application either to their client who has commissioned the application or to end users. Chapter 3.5 suggested that the tools that the mobile development

industry have built are geared towards this end with a minimum of extra effort for developers.

The word 'useful' is vague, and the concept of what embodies an application that might be useful either to someone commissioning an application or to an end user is unpredictable. To get a strong answer to the question of whether case can be used to build useful applications, therefore, needs case to be used in some of these situations: case needs to be used to build applications that have requirements imposed from outside.

It is worth noting that a positive answer to this question doesn't imply that the tool is widely usable; if one person can use case to build useful applications, then case can be used to build useful applications, even if the majority of developers find the concept mystifying. This is what distinguishes this question from the following question.

Can case as embodied in this tool be used by other developers? As noted, a positive answer to the previous question does not imply that developers in general can use case. This question, by contrast, is about the broader development community and about whether they can use case and whether it is useful to them. An answer to this question needs to come from the mouths of developers who have or have not found the tool useful.

This puts a constraint on the design of the tool. If developers are to comment on the usefulness of case, which is a novel element, then it is necessary for the rest of the tool to be as familiar as possible to them. If a developer is also struggling with other conceptual models and frameworks that are alien to their normal practice then the likelihood that they will be able to give any useful information about the use of case will be much reduced. This means that the design of the tool will have to follow the precedents of industrial tools as summarised in chapter 2, rather than the less-familiar precedents of the research tools.

3.11 Summary and conclusions

Case is a phenomenon in natural languages which allows for the categorisation of the relationships between words, and allows the order of words in sentences to change for contextual reasons or to fit the demands of a form. Case is found in languages from most of the major language families, on every continent that has its own languages, and, as far as evidence exists, at all points during the history of language. It is extremely unlikely that the similarities in case systems this far apart can be accounted for by chance, by borrowings and by genetic descent.

Not only do case systems exist in all these languages, but they are used to communicate similar meanings in languages the world over. Case is commonly applied to provide extra information to the speaker or listener about sentence structure; to express the recipients of transfers, the owners of objects; and for other meanings that are beyond the scope of this thesis.

In the book which reinvigorated the whole field of case research, Blake (2001) presented the case hierarchy. The case hierarchy is a typological model of case systems. It makes a statement about the kinds of case systems that are likely to exist and the kinds that are not, and what meanings are likely to be expressed through case. The case hierarchy will form the basis of the use of case throughout the remainder of this thesis. It has in general stood up well against new data.

Case, including the use of these frequently-expressed meanings could be useful in plastic user interfaces for one of two reasons. The first of these reasons is that it may be effective analogically: the meanings are widespread enough in different languages and widely known enough that developers may have them already in their heads as ideas that they can use. The second is that cases may tap into an underlying set of semantic roles which are not only a part of language. This second proposition is more controversial: some theories that include an explanation of case would agree that such a set of semantic roles exists, others would emphatically deny that it does. However, even if there is no such set of semantic roles, then the first reason might still stand.

To use case in plastic user interfaces consists of using cases to label the user's selections within the system. This can be used to attack the problem faced by mobile developers that was posed in chapter 2: it extends the industrial tools by providing a degree of dialogue plasticity while only introducing a small number of concepts (most case systems being fairly small) on top of developers' existing practices and tooling, unlike the research tools.

The use of one of the cases was found to be a productive way of examining differences in ordering assumptions between different platforms: a rule about behaviours of instruments or tools could be couched in terms of cases, and then this rule correctly described a difference in behaviour between two drawing programs on two different platforms. This analysis indicated two major weaknesses of a purely case-based dialogue modelling system. First, it is completely blind to dependencies: there is no absolute guarantee that two selections, the second of which depends on what the user selected for the first, might not have their order swapped. Secondly, if an interface is converted into another by means of applying reordering based on case, that reordering process may be lossy.

The evidence that case might be useful based on the exercise with the two drawing programs is backed up by the fact that there are parallels between definitions in the guidelines for iOS mobile applications and the common meanings

for the genitive and dative cases. In the light of this evidence the research questions presented in chapter 1 were re-examined and the shapes of answers to these questions were sketched out.

This chapter thus suggested that case might be useful for the creation of plastic user interfaces for mobile developers. The next chapter examines what 'useful' means in this context, and exactly how the usefulness of a case system can be assessed for this purpose.

Chapter 4
Methodology

4.1 Introduction

Chapters 2 and 3 argued for the possibility of a mobile development tool for plastic multi-platform user interfaces based on the notion of case, and argued that such a tool might be useful to developers. At the end of chapter 3, the research questions for the thesis were laid out. This chapter describes how these questions can be answered.

This is especially necessary for the last two research question, which are about the usefulness of case as embodied in the tool in building applications and the usefulness of case as embodied in the tool for developers. This chapter defines "usefulness" for the purposes of this thesis, and describes how this usefulness will be evaluated. How this evaluation is to take place must be considered before the tool is designed, because the tool needed to be amenable to being evaluated in the intended ways.

First, this chapter briefly discusses the act of building a tool to embody the principles laid out in the previous chapters, and summarises who the audience of this tool is. It is important that the tool is comprehensible to its intended users. It then presents three questions that were used to reflect upon the tool initially, immediately after its implementation. These questions are based on the implementation process and the structure of the tool. After this it briefly summarises the stakeholders in the process of developing mobile applications.

The chapter then describes the "3 Es" against which the tool was evaluated. These form the definition of 'usefulness' for the purposes of this thesis. The first E is 'effectiveness': whether the tool can be used to build high-quality applications and how the case system contributes to this. The second E is 'efficiency': whether the tool makes the developer more productive for a given amount of effort, and how case contributes to this. The third E is 'expressiveness': whether the tool allows the developer to build the applications that they want to build.

After this the chapter presents the three contexts in which the tool and its underlying notion were evaluated. The first of these was a case study type approach, where applications were designed for external users; the second was a workshop in which developers gave initial feedback from an intensive training and exploration session, and the third was a self-directed development exercise in which developers were given the tool to use for a longer period to design and build an application of their own choosing. These contexts emerge from the research questions.

The chapter then combines the three Es and the three contexts to demonstrate how the three contexts were used to shed light on the three Es. Finally, it summarises the methodology in the form of a table and in terms of which research question is answered by what part of the evaluation.

4.2 Building a tool

An abstract idea, in itself, is not something that can be directly used to build applications or something that can be given to developers or users to gather feedback and information. To get this kind of feedback for how well an idea actually works, different approaches to plastic interfaces have tended to be embodied in software tools, which are then evaluated.

The research tools examined in chapter 2 were all this type of tool. In addition, the Dialogue Flow Notation also outlined in chapter 2 was embedded into a software tool for evaluation (Book and Gruhn, 2004). The evaluation of the use of case in plastic interfaces for mobile applications will follow the same lines; the idea will be embedded into a tool, and that tool will then be used for the evaluation.

Similarly, to evaluate case, a tool was built, called "AppMaps". The design and implementation of this tool are outlined in chapter 5. This tool was then given to people for them to use and provide feedback about. It was designed (as specified by the exploration of the final research question in chapter 3) so that the major novel element would be the case system: every other design decision was aimed towards making the rest of the tool familiar. This was done so that the users would be more able to identify the contribution that the case system made to their use of the tool. The design of the tool itself also contributed to the answers to the first and second research questions, which are about the relevance of case to user interfaces and whether case is implementable.

Some immediate questions were posed about the tool itself during the design process. These questions were

Can a case-based approach be cleanly implemented? What it means for case to be implementable was presented in the second research question, in chapter 3. The tool itself can contribute a certain amount to the answer to this ques-

tion. The design of the tool will make it clear whether or not plausible meanings of cases line up with the user interface guidelines. If the tool proves to need many special cases to implement the user interface guidelines, then the implementation of case is not very clean, even if it is possible. If, however, plausible meanings for the cases can be implemented cleanly in terms of the user interface guidelines, then the implementation is clean.

How does the expressiveness of this approach compare to that of other tools? A certain amount about the expressiveness of the tool can be said simply from its structure. For this, Thevenin et al.'s (2003) Plastic User Interface snowflake, as summarised in chapter 2 was used. This provided a baseline comparison between the AppMaps tools and the other tools which were classified using this framework in chapter 2.

What are the limits on the tool's expressiveness? Some limitations on the possible expressiveness of the tool became evident immediately after it was built. If, for example, the way that the dialogue component was modelled made certain kinds of interface unable to be built with the AppMaps tool, then this was noted directly after the tool was built.

The answers to these questions are given in chapter 5.

The developer is not, however, the only stakeholder in the success of the system, and not the only person to whom the effects of case might be visible.

4.3 Who are the stakeholders?

Developers of all kinds, sometimes to their chagrin, do not work in glorious isolation. This is as true of mobile developers as of other developers. The other people who have an interest or a concern in the design of an application are the "stakeholders" in that application.

Exact definitions for "stakeholder" that are useful for this kind of endeavour are hard to come by. Sharp et al. (1999) give a number of definitions from both the literature about organisations and their impact on society and the literature about information systems and their impacts on people. They conclude that these definitions are not useful for the purpose of actually finding out who the stakeholders are in any given situation, especially in the early part of the process of building a system.

Sharp et al. remedy this by giving a method to find the stakeholders in a system by starting with a set of known "baseline stakeholders" and working outwards. This method is recursive, and has no clear termination condition. They leave this up to common sense; indeed, they say in the conclusions to their paper that "knowing when to stop looking is as important as knowing when to look" (p. 390). In

the case of evaluating case for mobile applications, simply using the baseline stakeholders is sufficient, because beyond this one begins to find only people who are not involved with the application directly, and thus whose interest in how it was built is minimal.

The baseline stakeholders that Sharp et al. identify are "users", "developers", "decision-makers" and "legislators".

USERS are "the people, groups or companies who will interact with the software and control it directly, and those who will use the products (information, results etc) of the system" (p. 389)

DEVELOPERS include "analysts, designers, programmers, testers, quality assurance representatives, maintainers, trainers, project managers and so on." (p. 390). These are the people who are involved in the technical aspects of building the system.

DECISION-MAKERS are those responsible for managerial oversight. Examples of this kind of stakeholder are "managers of the development team, user managers and financial controllers in both the developer and the user organisation" (p. 389).

LEGISLATORS are those who set the rules by which the system must abide. This includes, but is not restricted to, legislators in the governmental sense; the category includes "professional bodies, government agencies, trade unions, legal representatives, safety executives, quality assurance auditors and so on" (p. 389).

For the purposes of this evaluation, the process is being looked at from the perspective of the developer developing the mobile application, and so the baseline stakeholders are:

USERS are those who will use the mobile applications produced with the tool.

DEVELOPERS are those who will use the model to build a mobile application. These are the immediate users of the tool that embodies case.

DECISION-MAKERS are those who will have the final approval of the application. In cases where a small company or an individual is developing an application, then most of the decision-making power will rest in their customer.

LEGISLATORS are the companies that can make demands on developers about the user interfaces of the applications; specifically those who set the user interface guidelines that developers have to adhere to in order to get their applications published.

4.3.1 Application users

The relationship between application users and the development tools is a murky one. The precise nature of the tool that has been used to create an application is generally not visible to the user of the application: and two applications that adhere equally to the user interface guidelines for the platform could have been created in using entirely different tools.

Therefore, for the purposes of this evaluation, the user's wishes were considered to be taken into account by adherence to the style guide.

4.3.2 Developers

As noted above, the immediate users of the tool are developers. It is they who will be sitting in front of the tool and using it: it is they who will need to understand it in detail and understand how to use it in order to create functional applications. Therefore, it is their expectations that must be respected in the design of the tool.

However, evaluating developers comes with a complication. Developers are expert users, and their tools are the tools of expert users. To evaluate a development tool fully requires it to be used by expert users *of that tool*: if a developer is presented with a new tool, it will take them some time to get to a position of expertise in the use of that tool.

Since AppMaps is a novel tool, there was only one expert—the author. However, the existence of this thesis demonstrates in itself that the author may not be a typical developer. Therefore, the experiences both of the author and of other developers needed to be taken into account.

4.3.3 Commissioning customers

The person or people who commission the application are not direct users of the development tools. However, it is often their ideas about functionality and user interfaces that the developer must turn into working applications.

If their ideas, or the functionality that they want, cannot be implemented using a tool but could be implemented with another tool, then this potentially shows a weakness in the first tool.

Therefore, the experiences of commissioning users needed to be taken into account.

4.3.4 Platform owners

As laid out in chapter 2, the companies that develop mobile platforms tend to retain a great deal of control over what software runs on those platforms.

It was not feasible to involve the manufacturers of mobile companies directly in the evaluation of the tool. Fortunately, however, their wishes for their platforms are made available to developers in the user interface guidelines. Thus, evaluating against the user interface guidelines of a given platform also took into account the wishes of the mobile platform owners.

4.4 The three Es

Having established the roles in the evaluation process, it is necessary to outline the criteria that will be used for evaluation. Chris Greenhalgh (private communication, 2012) suggested a framework for evaluating the usefulness of development frameworks based on three criteria: effectiveness, efficiency and expressiveness.

4.4.1 Effectiveness

The word 'effective' as used here means that the framework can be used to produce user interfaces that are of high quality. All of the stakeholders have an interest in the quality of the user interface. The developer has an immediate interest in providing a quality application to their commissioning user (should they have one) or directly to the world (if they do not). The commissioning person has an interest in receiving a good quality product from the developer. The user wants an application they can actually use. The developer of the mobile platforms wants applications that will not bring their platform into disrepute.

In this thesis, the conception of 'quality' is based on the opinions of the stakeholders. As noted above, the user's perceptions and those of the developer of the platform are taken into account by the use of the user interface guidelines.

4.4.2 Efficiency

The term 'efficient' here means that the use of case enables developers to be more productive for a given amount of effort than the tools they already have allow them to be. 'More productive' is a vague term: this thesis mostly concentrates on the subjective experiences of developers.

This is because actually measuring developer productivity is fraught with issues, largely due to the very large number of variables involved in the development process. Nelson (1967), in an early study for the United States Air Force, identified 99 distinct variables in the environment that impacted on the efficiency of programmers (pp. 54–59) and many more which impacted on the efficiency of the software engineering process as a whole (throughout the rest of the document). A more recent analysis by Jones (2000) gives around 250 variables that he claims affect programmer performance.

The stakeholders who have a direct interest in the efficiency of developers are the developers themselves and their commissioning people: the former because it directly impacts on their time, and the latter because they have an interest in getting the application finished. The interests of the users and the companies that develop the platforms are much less involved.

4.4.3 Expressiveness

The word 'expressiveness' is here to do with the range of meanings that the tool can encompass. The tool is expressive if it lets the developer create an application for the problem domain and platforms that they want to create for. It is distinct from 'effectiveness' because it is by no means impossible that a tool could concentrate on building very high quality user interfaces in a very restricted range of areas; or that a tool could be broadly applicable but build very poor quality interfaces.

The stakeholders that have a direct interest in the expressiveness of the tool are the developer and their commissioning people. The end user does not come into direct contact with the tool and is likely not even to know what tool was used to build the application they use; and the developers of the mobile platform have limited interest in third-party tooling.

4.5 Contexts of evaluation

The tool and its case system were evaluated in three basic contexts, each of which provided a different kind of information about the development process and take a different subset of stakeholders into consideration. The three contexts of evaluation are not controversial; each has been used elsewhere, and a similar three in combination were used to evaluate the MARIAE tool (Paternò et al., 2011).

4.5.1 Case studies

Three applications were developed by the author as case studies. In each case study, a company commissioned the author to develop an application. The three companies were in very different market sectors and required very different applications. The results from the case studies are given in chapter 6.

The process of evaluation through case studies is not unique to this thesis. ARTStudio was evaluated by building an application for the EDF energy company to control heating in a house (Thevenin et al., 2003). Multimodal TERESA was demonstrated as being useful through a series of short case studies (Paternò et al., 2008). DiaGen was also evaluated through a small case study, implementing the process of registering for a website (Book et al., 2006). As noted above, the process of evaluation for MARIAE included case studies.

The case studies take all the stakeholders above into account. The developer is the author who, as noted above, was the only expert user of the tool. The commissioning customer was the person inside the company that was the author's point of contact with the application. As noted above, the user's interests and those of the companies that control the platforms were assumed to be encapsulated in the relevant user interface guidelines.

The case studies are primarily useful for providing information for the third research question, about whether case can be used to build useful applications.

4.5.2 Developer workshop

To capture initial developer feedback on the tool, seven mobile developers came to a workshop. In this workshop, they worked through the development process of two example applications. The first of these was a guided activity to teach them how to use the tool: the second was an activity where they were given a specification from a hypothetical customer and asked to build an application. In both cases the source code for the application had already been written: their interaction with the system was limited to putting the application together from components, so that they only interacted with the dialogue controller and the case system.

Evaluation of development tools by workshops and self-directed development by developers is not controversial. Of the tools in chapter 2 UIML was evaluated in a workshop situation (Phanouriou, 2000, ch. 7) and released for other developers to use, extend and comment upon (see, for example, Binnig and Schmidt, 2002, and Luyten and Coninx, 2005). Multimodal TERESA was also evaluated in a workshop event, although the details of the methodology and results are contained within an EU project document that is not, at time of writing, publically accessible. As noted above, the MARIAE evaluation contained a developer workshop.

The results from this workshop only took the developers' views into account. None of the other stakeholders were represented. It provided information about a very specific set of interactions between the developers and the tool. It also provided information about the quality of the interface of the tool itself that allowed it to be slightly refined for the following study. The model and mode of operation of the tool remained unchanged, however. The majority of the information from this scenario feeds into answering the fourth research question, about the usefulness of case to other developers.

4.5.3 Self-directed development

The third and last study was a longer study also involving external developers. In this study, seven mobile developers were given the revised tool and asked to build an application with it. No constraints were put on what applications they could develop. If they wished to, they could develop an application for a commissioning customer.

As noted above, the evaluation of UIML has proceeded down this route almost by default, as developers have picked up the toolkit and written renderers for it and experimented with it in different situations. The evaluation of MARIAE also used a longer study, although there the developers had a problem set for them, rather than being able to choose their own application to build.

The results from this self-directed development study took into account the points of view of all stakeholders except for commissioning customers: while the developers were free to build applications for commissioning customers if they wished to, none of them did this. The majority of the information from this scenario feeds into answering the fourth research question, about the usefulness of case to other developers.

4.6 The three Es in the three scenarios

Each of the three Es can be considered in each of the three scenarios. This section presents the questions that will be used to structure this part of the evaluation.

4.6.1 Effectiveness

In the case studies, two areas were examined to understand the effectiveness of the tool.

Was the commissioning user happy with the application on all platforms and form factors? Free-form criticism was gathered from commissioning users about the application, started by asking them specifically what they liked about the application and the development process, what they disliked about the application and the development process, and how close the application was to the application that they had actually wanted. They were also asked to concentrate on the parts of the application that case had contributed to.

Did case contribute to the application's conforming to the human interface guidelines for the desired platforms and form factors? Each part of the application that case contributed to was examined against the user interface guidelines of the relevant platforms. Also, case could not be expected to cover the whole of a set

of user interface guidelines: therefore, the proportion of the user interface guidelines that were covered by case was also assessed.

In the workshop, four areas were examined:

Were developers happy with what they had produced? During a semi-structured discussion session at the end of the workshop, developers were asked to criticise the quality of the applications that they had produced during the workshop. They were especially asked to criticise the parts of the application that case had been applied to.

Did their solutions resemble a sane model answer? For the activity where the developers built an application following a specification, there was a model answer that fulfilled that specification. In situations where the developer's solution deviated from this model, they were asked why they had chosen the solution that they had, again specifically with regards to case.

Where did they feel the tool fitted into the problem of plasticity? As stated above, case cannot be expected to cover the whole of a set of user interface guidelines; nor can it be expected to cover the entire problem space of plasticity. Developers were asked during the discussion session to comment on the extent to which case addressed this problem space.

The problem space here is not meant in the sense of the plastic user interface snowflake (as in chapter 2). Instead, it is meant in the sense of the problems that they face in their professional practice while building plastic user interfaces.

Would they find the tool useful in their professional practice? During the discussion session developers were asked whether a case-based approach would be useful in their professional practice and whether it would allow them to produce plastic user interfaces of a high quality.

In the self-directed development study, three areas were considered, two of which are analogous to the areas in the case studies.

Were developers happy with what they had produced? At the end of the application development process, developers discussed their applications either face-to-face, by telephone, or by email. They were asked to criticise the quality of what they had produced with the tool compared to what they could have produced using other tools, and to note whether the results met their standards as professional developers.

Did case contribute to the application's conforming to the human interface guidelines for the desired platforms and form factors? The applications that developers had built were gathered in at the end of the development process. The parts of each application that were affected by case were examined for compliance with the user interface guidelines of relevant platforms, not by the developer of the application, but by the author.

By these means, the effectiveness criterion gathers information about the *perceived* quality of the parts of the interface produced by the case system. It gathers this information from the developer and from the commissioning user. It also measures slightly less subjective information about the quality of these pats of the interface against the user interface guidelines that, as remarked above, stand as a proxy for the wishes of both the user and the platform owner.

4.6.2 Efficiency

In the case studies, two areas were considered for the evaluation of efficiency.

How many software components are involved in effecting plasticity? Is case responsible for any reduction compared to the other comparable industrial tools? A software component is something that is either a class or singleton object or something that becomes a class or singleton object at run-time. For example, a class that implements a view on some data is a software component; a class that implements the storage of that data is a software component; a stylesheet is a software component. This question will be answered by counting the number of these software components.

This is an important measurement because it measures how well the concerns of plasticity are separated from the other concerns in the application, and how well plasticity is encapsulated.

Consider the problem of plasticising the dialogue layer of an application. In a tool that had no awareness of the dialogue as a separate software component, each view that wanted to avail itself of dialogue plasticity would have to implement that plasticity itself. In this extreme case, therefore, the number of software components that effect dialogue plasticity would be linear in the number of places in the interface that this plasticity is called for. This, in turn, leads to a higher maintenance load on the developer.

As a concrete example of this, from 2010 to 2014, the author was involved in the building of a satellite navigation system for the canal system of the United Kingdom that ran on mobile devices. The development of this application was done entirely using orthodox industrial cross-platform techniques. To work well on tablets, the tool needed to have some adaptations

made to the user's dialogue with the machine. At every point where this adaptation needed to occur, a change had to be made to the source code that was implementing those views: and every adaptation occurred in a different view, so that the concern of adaptation was spread all over the source code for the application. In this case, the number of software components that implemented this plasticity was equal to the number of places in the application that the dialogue had to be adapted

By contrast, consider an ideal tool that completely encapsulated the problem of dialogue plasticity. In this tool, it would not matter in how many places plasticity was required: the number of software components that were involved in effecting that adaptation would remain constant.

Case certainly cannot be expected to be an ideal tool: however, whether the number of software components that effect plasticity is linear in the number of uses of case or whether it is constant *in common use cases* forms an argument that, in those common cases, it makes the developer more efficient.

Does the use of case provide a tangible benefit in terms of time at any specific point in the development process? Since the applications were of different sizes and complexities and for different commissioning users who worked in different ways and at different speeds, an absolute measurement of time taken to perform activities would not have been useful. A more useful piece of information would be what, if any, development activities were noticeably faster than they would have been otherwise.

In the workshop, two areas were considered.

Would this tool make the development process more efficient, and if so, which parts of it? This criterion is not merely subjective but also subjunctive: it relied not only on the developer's experience but also on their ability to hypothesise accurately about their use of the tool. This means that information gathered in this area cannot, on its own, be weighted very heavily.

However, what it does provide is context for evidence for efficiency from the other two evaluation activities. Agreement between the the speculative comments of the developers in the workshop and the experiences of the developers building tools makes the evidence that case increases efficiency in those areas stronger.

Would the case system of the tool be worth learning for the expected efficiency gains? An efficiency gain that requires the developer to do so much learning that it cancels out the time and productivity gained is not a real efficiency gain at all. To evaluate whether this was the situation with case, the developers at the

workshop were asked whether they felt that the overhead in learning about the case system would be justified given the efficiency gains they expected from the tool.

This, again, relies on both the developers' self-awareness and their ability to accurately hypothesise about their own professional practice. Again, though, when combined with information from the other two evaluation activities it provides context.

In the self-directed development study, three areas were evaluated.

How many software components are involved in effecting plasticity? The same analysis of software components was done on the applications generated by the self-directed development study as on the applications generated from the case studies. Again, this was to evaluate whether the tool provided efficiency gains by encapsulating the concerns of plasticity that the developer was likely to run into in normal use.

Does the use of case provide a tangible benefit in terms of time at any specific point in the development process? This, again, mirrored the evaluation of the case study applications to ensure that any results from those were not due to the peculiarity of the author.

Would the case system of the tool be worth learning for the efficiency gains experienced? This mirrors the evaluation from the workshop, but provides more solid evidence. After their use of the tool, developers were asked whether the efficiency gains they had seen in the design of their application, if any, had made it worth learning to use the case system of the tool.

4.6.3 Expressiveness

The case studies evaluated the expressiveness of the case system in three areas.

Does it cope well with the platforms and form factors that the commissioning user needs? As noted in chapter 2, the list of major mobile platforms is ephemeral and prone to rapid change. If, however, a platform that the commissioning user needs is missing from what the tool can do, then that needs examining, because there are two possibilities as to why this could be the case.

Firstly, it could simply be a vagary of the implementation. In this case, the ideal would be to port the tool to the platform required to demonstrate the applicability of case to the platform.

Secondly, it could show up a limit of the model itself. In this case it would not be trivial to port the tool, even if all the technical factors were easy: and

the model of the resulting tool would be noticeably different from the model of the original tool.

Therefore, under this area of evaluation, the absence of problems would have been somewhat indicative of broad applicability across mobile platforms in general. A more interesting and definite result would have been a failure of the tool in one or more platforms for the second reason above, as this would have set definite limits on the applicability of case.

What kinds of applications can case be useful for? People use their mobile devices for a large number of purposes, which means that multiple kinds of application exist. For example, some applications primarily exist to help the user browse a data set or find answers from data: examples of this kind of application include satellite navigation applications, library catalogues and interactive dictionaries. Some applications exist to facilitate communication between people: examples of this kind of application include instant messaging applications and videoconferencing applications. Still other applications exist to allow the user to consume multimedia content: examples of this are film rental applications and internet radio applications.

Many applications, of course, do not sit in one of these categories. Email applications both provide the user with the ability to browse the data set of their own email and to communicate with other people through email. Applications such as Google's YouTube allow the user both to consume multimedia content and to communicate with other users about that content. Therefore, this kind of division of applications into categories should be taken as a rather imprecise and analogue division, rather than as a set of absolute categories.

For the purposes of this thesis, no canonical set of kinds of application were defined in advance: different developers may categorise their applications different ways, and a pre-defined categorisation risks losing unexpected insights into the development process.

In the case studies, the applications produced were divided according to their primary function.

Is case more useful for some industries than others? Mobile devices have been found to be useful in many kinds of human industry. A startling range of problem spaces are represented in the application stores of the major platforms.

Case is not specifically tied to any one area of human endeavour: it acts as a semantic framework into which, in theory, any nouns can be placed. However, there may be some hidden assumptions built either into case itself

or into the specific case system used in the tool. If this is the case then it may be more useful for some industries than others.

In the workshop, three closely-related areas were evaluated:

Are there platforms or form factors to which this tool is not applicable? This question mirrors the first question in the expressiveness evaluation of the case studies. Developers were asked to comment on whether there were platforms or form factors to which the tool was unsuited, drawing on their own professional practice and that of their peer group.

Are there kinds of applications to which this tool is not applicable? This question mirrors the second question in the expressiveness evaluation of the case studies. Developers were asked to comment on what kinds of application the tool was suited to and was not suited to: and, if there were areas where the tool was not applicable, how much of an impact would this have on the applicability of the tool in their professional practice.

Are there industries to which this tool is not applicable? This question mirrors the third question in the expressiveness evaluation of the case studies. Again, developers were asked to comment on this based on their own professional practice and that of their peer group.

In the self-directed development study, a similar three areas were evaluated:

Does case work well with the platforms the developer used? This mirrors the first question in both the expressiveness evaluations above. It is distinct from the case study, though, because the judgements here are further removed from the author's: and it is distinct from the workshop because it relies on the concrete work that the developer did during the self-directed development study rather than their speculation and hypothesising based on their professional practice.

Does case work well in the kinds of application the developer created? This mirrors the second area in both the expressiveness evaluations above. It is distinct from the case study, again, because the judgements are not the author's or the author's customer's. It is distinct from the workshop because it is based on the concrete work that the developer did during the self-directed development study rather than developers' speculation.

There is another distinction that makes this question a valuable addition to the workshop study's parallel question. In the workshop study, the developers were all in the same room, and the vocabulary they used to discuss

the kinds of applications they were building would therefore be a negotiated part of the conversation in that room, so if individual developers had more idiosyncratic ways of dividing applications into kinds, then those might not show. By contrast, each developer in the self-directed development study was working alone: so if their division of applications into kinds agreed with the other developers' and with the workshop participants', then this suggests that these kinds of application form a genuine basis to compare the feedback from developers.

Does case work well in the industries that the developer was working in? This mirrors the third area in the expressiveness evaluations above

4.7 Summary

The above is summarised in table 4.1. This table concisely lays out the structure of the evaluation: corresponding tables in chapters 6 and 6 present the results of the enquiries. Each of the research questions then can draw its answers from a subset of this information.

Are the categories delineated by case relevant to user interfaces? In chapter 3, this question was said to require evidence of three things: that the vocabulary of case is applicable to user interfaces; that the system follows the rules of case sufficiently well to be referred to as case; and that this vocabulary is amenable to unexpected uses by developers.

Information on the first of these can be gathered from the expressiveness information gathered from developers and from the case studies; if developers can use case naturally and find it relevant to user interfaces, then a positive answer to this question is warranted.

Information on the second of these must be gathered from the implementation of the tool itself (presented in chapter 6 below). The constraints imposed by the case hierarchy need to be taken into account during the building of the tool.

Information on the third can be gathered from the case studies and the longer development studies. The design of the tool (outlined in chapter 6) and the documentation given to the developers will provide examples. If the case studies contain uses of case that are valid but unexpected according to the design of the tool, or if the developers use cases in ways that meet the definitions of the cases but are not the ways that the cases were used in the documentation, then case has a claim to be a living category.

Case studies		
Effectiveness	Efficiency	Expressiveness
Was commissioning user happy?	Number of software components involved.	Did it meet platform and form factor requirements?
Evaluation against human interface guidelines.	At what points did case provide a tangible benefit?	What kinds of application suit case?
		Is the use of case industry-specific?
Workshop		
Effectiveness	Efficiency	Expressiveness
Were developers happy?	Would the case system make developers more efficient?	Are there platforms or form factors to which case is unsuited?
Comparison with model answer.	Would the case system be worth learning?	What kinds of application suit case?
Where did the tool fit the problem?		Is the use of case industry-specific?
Would the tool be useful to them?		
Self-directed development		
Effectiveness	Efficiency	Expressiveness
Were developers happy?	Number of software components involved.	Did it meet developer's platform and form factor requirements?
Were commissioning users happy?	At what points did case provide a tangible benefit?	What kinds of application suit case?
Evaluation against human interface guidelines	Would the case system be worth learning?	Is the use of case industry-specific?

Table 4.1: Summary of methodology

Can a tool be built that embodies case for building plastic user interfaces? The answer to this question must be settled by the actual implementation—or a failure to implement—such a tool. In chapter 3, this question was split into two parts: first, whether such a tool is technically implementable, and secondly whether such a tool produces sensible results.

The first part of this question warrants a positive answer if the definitions of the cases can be turned into software. An answer to this will emerge from the attempt to build such a tool, in chapter 6. An answer to the second part will emerge from the information gathered on whether the use of case produces applications that conform to the user interface guidelines for platforms, under the effectiveness criterion of both the case studies and the longer, self-directed study.

Can case as embodied in this tool be used to build useful applications? The primary source for information for this will be the case studies and the self-directed development study outlined above, evaluated in terms of the three Es.

Can case as embodied in this tool be used by other developers? The primary sources for information for an answer to this question will be the workshop and the longer, self-directed development studies, both evaluated in terms of the three Es.

The next chapter begins the process of answering these questions by laying out the design decisions that were made in the creation of the tool and demonstrating how the design of the tool emerged from the structure of case systems and from the requirements placed on it in this chapter.

Chapter 5

Design and implementation

5.1 Introduction

Chapter 4 outlined a way of evaluating the usefulness of case for building plastic user interfaces for mobile applications. The first part of that evaluation involves building a tool to encapsulate the idea of case and make it available to developers. This chapter describes that tool, which is called "AppMaps". The goal of the AppMaps tool was to provide a useful degree of plasticity to industrial developers for building applications for Android and iOS using the idea of case. The goal of this chapter is to demonstrate how the structure of case systems informed the design of the tool, and how the plasticity effected by the tool emerges from its case system. This tool targets iOS and Android phones and tablets, making a total of four targets. As noted in the previous chapter, the tool should be as familiar as possible to developers. Therefore, the tool is based on Sencha Touch, which was the most popular cross-platform tool at the time the Vision Mobile report cited in chapter 2 was written. Each other component of the software takes its cue from industrial tools that are in use for building cross-platform mobile applications.

The tool is described as a series of layers, starting from the abstract idea of case and working outwards to a tool that can be used to build mobile applications. The innermost, essential layer is the design of a case system for the tool. This is described in section 5.3 below. The next layer out is how those cases should be realised in actual concrete user interfaces (section 5.3.4). The next layer is that of a dialogue controller and stylesheet mechanism that uses case to realise the concrete user interface patterns (sections 5.4 and 5.5): and the outermost layer is a software architecture and an implementation of that architecture that can be used to build applications (section 5.6).

After these layers are described, the chapter summarises the lifecycle of an AppMaps application, showing how the layers interact to effect a degree of plasticity. This lifecycle stretches from the moment the developer presses the "build" button to the moment that the application is shut down on a mobile device.

Chapter 4 noted some questions about the tool's expressiveness that could be answered immediately after the development of the tool. At the end of this chapter, these questions are answered.

5.2 Accidents of implementation

In several places in his work, Aristotle talked about a distinction between essential and accidental properties of things, starting a hare which has occupied scholars ever since. One way of classifying properties into essential and accidental properties is modal (Robertson and Atkins, 2013): an essential property of a thing is one that it must have in order to be that thing, whereas an accidental property is one that it just happens to have and is not key to its identity.

The essential property of AppMaps, the property that makes it a way of answering the research questions asked, is the fact that it embodies a case system. A case system alone does not a software engineering tool make: and so other parts of the system had to be built. However, these were not essential: they could have been built a number of other ways and still produced a tool that could have answered the research questions. For example, AppMaps does not use a model-driven engineering approach, but a model-driven tool using cases could have been used to answer the research questions. In the case of AppMaps, the accidental properties of the system were generally arranged to make it familiar to industrial developers of mobile applications.

This chapter will concentrate on describing the essential properties of the tool, that is to say its case system. Accidental properties will be discussed briefly where they will play a role in the discussion in future chapters.

5.3 The case system

5.3.1 Application of the case hierarchy

The overall structure of the case system is defined by the case hierarchy. The case hierarchy was summarised in chapter 3 and is diagrammed again here in table 5.1 for reference.

Recall that natural languages tend to have case systems anchored to the top of this list. So a case system with two cases will tend to have a nominative and an accusative or an absolutive and an ergative (these cases were described in section 3.5.1). A case system with three elements will tend to have these two cases plus a genitive: and so forth.

In the current context, however, the situation is a bit more complicated. In section 3.10, it was noted that not all the meanings of cases can be applicable to

Nominative
Accusative or Ergative
Genitive
Dative
Locative
Ablative or Instrumental
Other cases

Table 5.1: Blake (2001)'s case hierarchy

user interfaces. Specifically, the nominative and accusative or ergative and absolutive are specific to the kinds of sentence structure one gets in natural languages. Therefore, the case system of the tool begins with the genitive and works downwards.

5.3.2 Structure of the case system

The AppMaps case system is a three-element system. Its two specialised cases are a dative and a genitive. The third, being the lowest on the case hierarchy, gets everything else rolled into it. Cases that have a lot of different functions are often called oblique cases in the literature (although this terminology is a little confused; see Nichols, 1983). Therefore, this case will be referred to in the remainder of the chapter as the oblique case.

5.3.3 Meanings of the cases

Each case in the AppMaps tool has a range of meanings. Blake's case hierarchy deals in very abstracted, stripped-down definitions of what each case is. However, in actual languages, as shown in chapter 3, cases do not obey strict clean divisions between them and between the functions and meanings they have. To an extent, then, allocating meanings to the cases in AppMaps is an act of construction which must be guided by the meanings of cases in natural languages.

The meanings of the cases are shown below. Comparing these with the examples given in chapter 3 will show that they correlate well with meanings found in natural languages.

The genitive is used for ownership or origination. If the user selects somebody who owns something or from whom something comes, then selects a something that is owned by or comes from that somebody, then that somebody should be marked as genitive. For example, in an instant messaging application, if the user can browse messages by sender, then that sender could be marked as a genitive.

It is also used for categories, or for qualities or predicates, if things that have that quality or obey that predicate are being browsed. For example, in a satellite navigation application that provides the user the ability to filter based on kind of point, then the user's selection of the kind of point could be marked in the genitive. Likewise, in an email application that allows the user to browse their inbox by folder, the user's selection of the folder could be marked as being in the genitive.

It is also used, as an extension of this, with a partitive meaning: the whole of which a part is being manipulated or looked at can also be marked as genitive. For example, if the user were browsing a list of documents and looking at their contents, the list of documents could also be marked as being in the genitive.

The dative is used for a beneficiary or for the destination of an act of sending. For example, when the user selects a destination to share a piece of content to, that should be marked as being in the dative: the options in the share panel (which at present usually include Twitter, Facebook and a slew of other sharing destinations) could all be marked as being in the dative.

It is also used for situations where the user needs to be able to provide more open-ended information about a destination. For example, if the user needs to enter an email address for the destination of an email message, or the username of a person in an instant messaging application as the destination of a message, this too should be in the dative.

Note that it is *not* used for the spatial destination of a movement: it should not be used, for example, for the end point of a route planning operation, or for the spatial destination of the movement of a robot.

The oblique is used in situations when neither of the other two cases are appropriate.

5.3.4 User interface patterns

Each case is realised in certain ways in the final user interfaces of applications. This section summarises how the cases can be realised within the user interface guidelines of each of the platforms that AppMaps targets.

The genitive has similar, but distinct, realisations on iOS and on Android. The iOS human interface guidelines and precedent are quite clear; the Android ones far less so.

On iOS tablets, genitives are displayed as split views (figures 5.1, 5.2 and 5.3). The genitive selection is in the left hand pane: the objects that belong or are

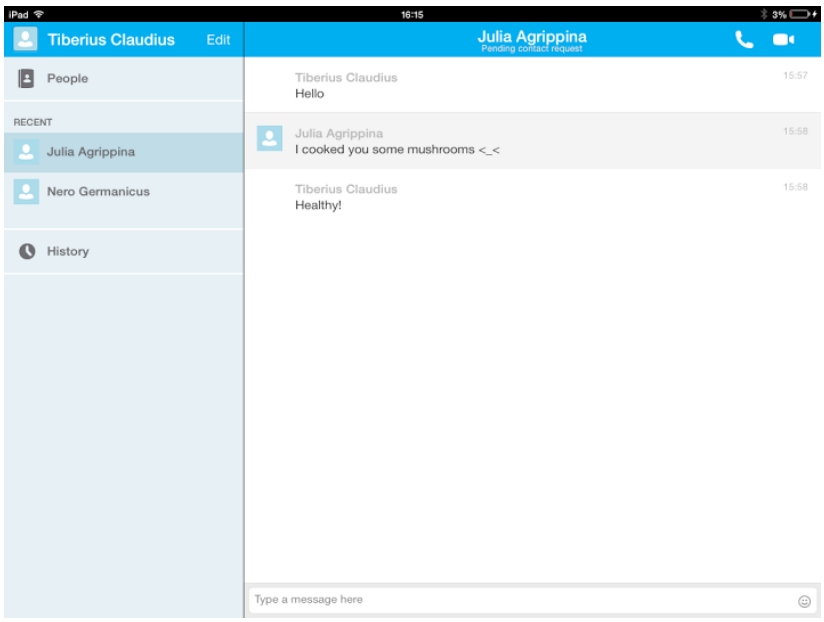


Figure 5.1: Genitive of ownership on iOS tablets

in the category are in the right pane. In the first, the owner or originator of messages is selected on the right hand side and the message is selected on the second; in the second, a category of settings is selected on the left and the individual settings on the right; in the third examples an item (either a person or an email) is being selected in the left hand pane and the contents or details of that person are shown on the right.

On iOS phones, genitives are displayed as two screens: the user first makes the genitive selection, and then the object from the person or category, or views the details. This is shown in figures 5.4, 5.5 and 5.6. Each of these figures is of the same applications as its counterparts in figures 5.1, 5.2 and 5.3 above. When both the genitive object and the other object are being selected from list-like views, each item in the first list shows an arrow (as in figure 5.5).

On Android tablets, the user interface guidelines are less prescriptive and so more freedom is left to the individual application designer. However, a similar pattern is frequent and permitted by the user interface guidelines. Examples that parallel the iOS examples are in figures 5.7 and 5.8.

On Android phones, there are two patterns for genitives. Which pattern is in use depends upon the age of the application and the developer's aesthetic preferences. Until recent versions of Android, the pattern was the same as the iOS phone pattern: the user first selected an owner or category or object

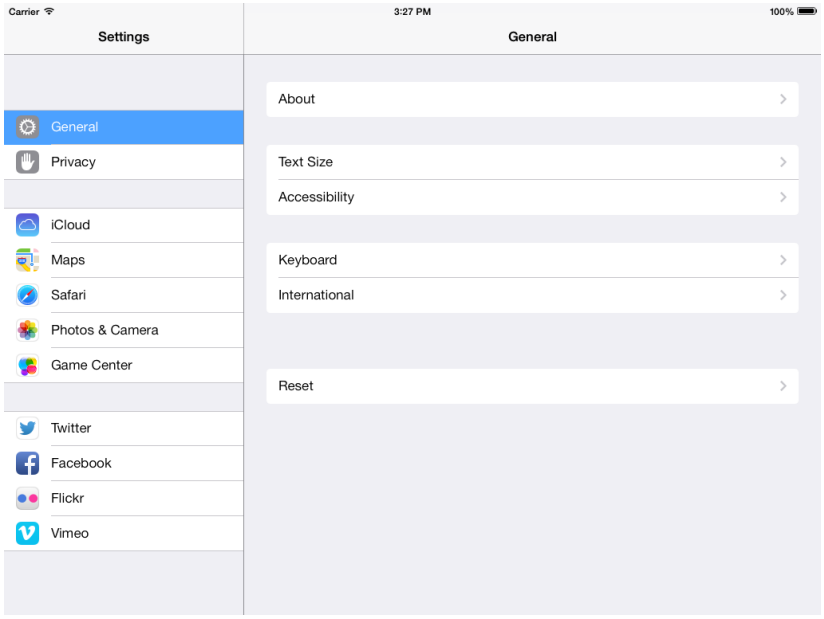


Figure 5.2: Genitive of category on iOS tablets

and then the objects in the category or details of the object were displayed. Recently, since Android 4, a second pattern has been suggested by the user interface guidelines. In this case, the objects are displayed first, displaying the last category the user chose or a sensible default: a button is then provided in the top left of the screen to display the list of categories. The list of categories slides out from the left of the screen. AppMaps implements only the first of these, because the development of the tool was already finished by the time the revised Android user interface guidelines were published. Examples of genitives on Android phones corresponding to the examples above for tablets are shown in figures , 5.9 and 5.10.

The dative has two realisations on each platform: one for situations where the user has to choose between a number of pre-defined destinations, where each destination appears as a button-like control, and one for situations where the user is entering an address or a phone number, where the box for the user to enter the destination is not button-like.

On iOS tablets, the first is implemented as a "popover" (figure 5.11). A popover is a temporary view that appears over the content being browsed, with a "speech bubble"-like tail pointing to the control that summoned it. In this popover, the icons for the destinations are displayed. When the user taps outside the popover the popover is dismissed. In native applications, this popover is managed by an "activity view controller".

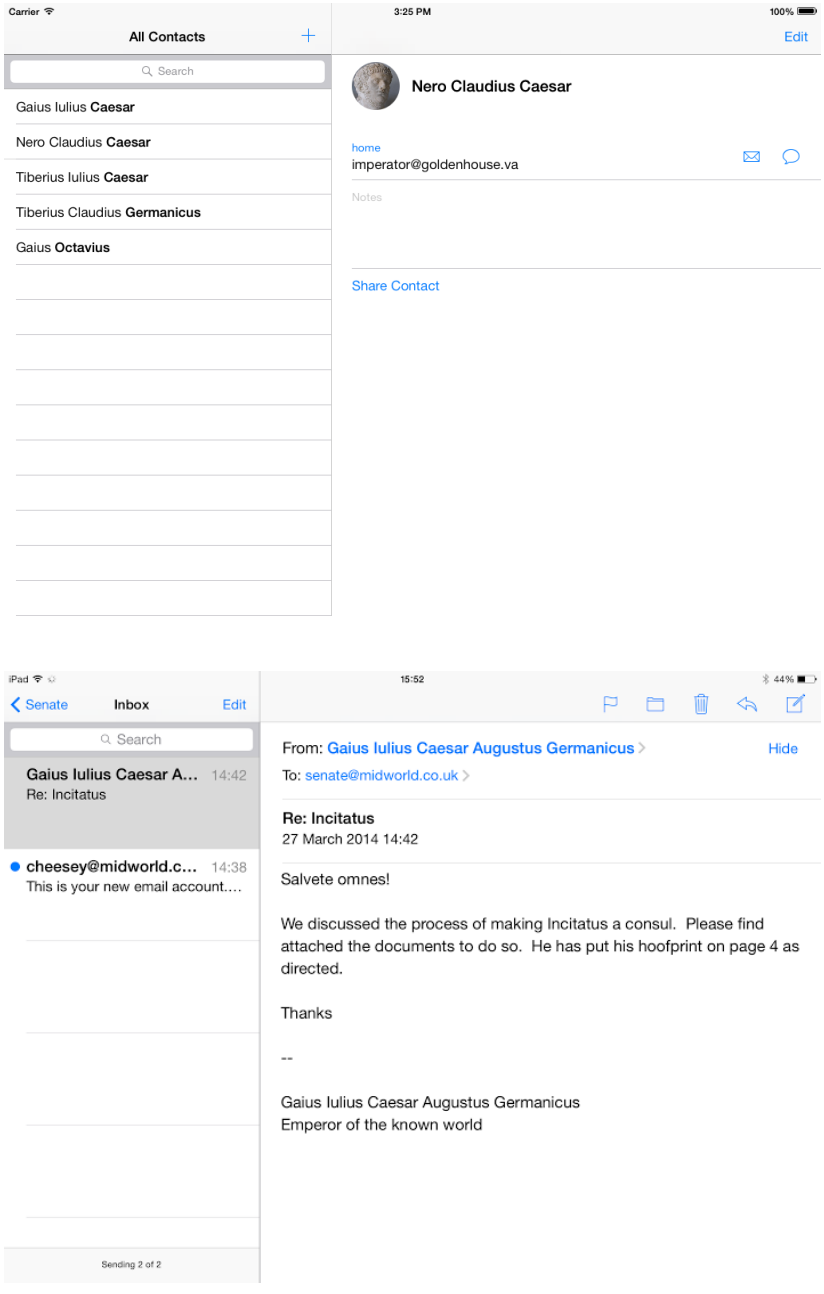


Figure 5.3: Partitive genitives on iOS tablets

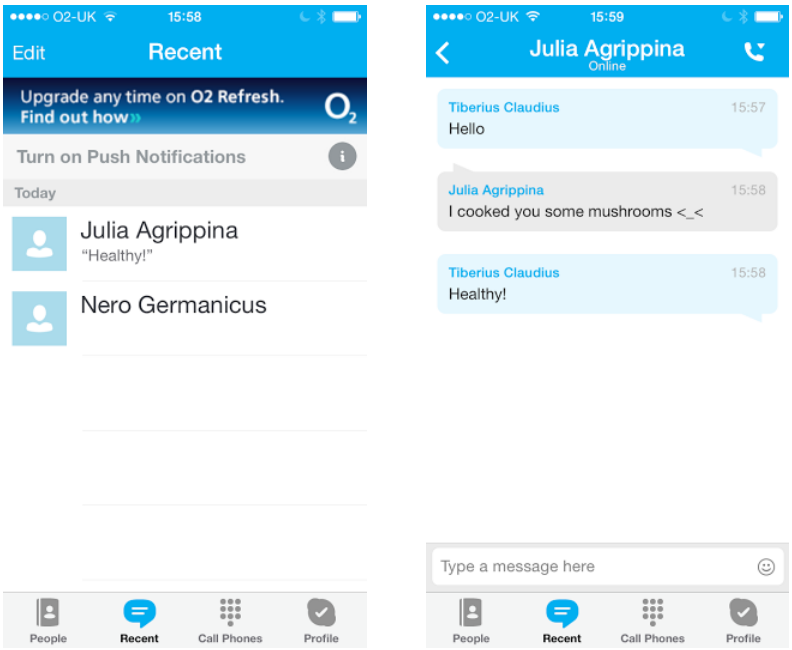


Figure 5.4: Genitive of ownership on iOS phones

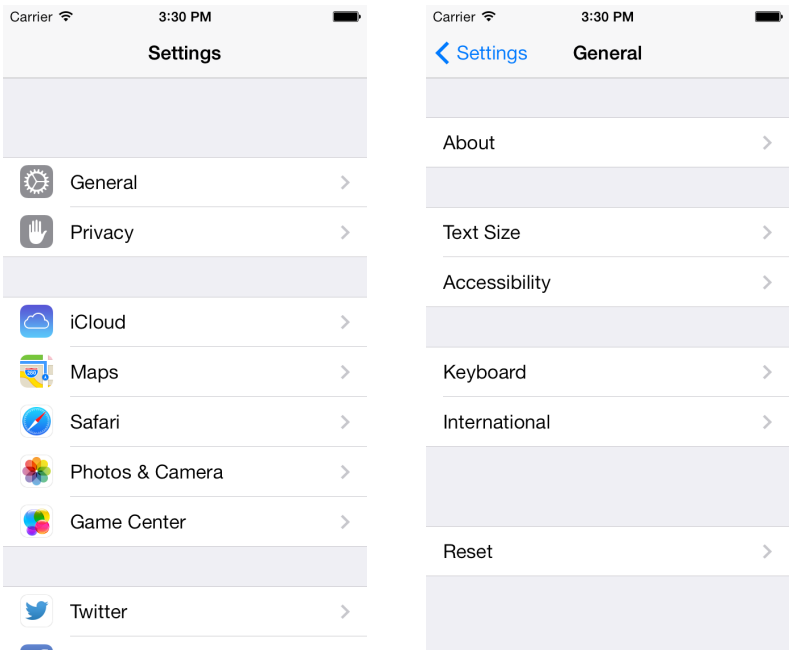


Figure 5.5: Genitive of category on iOS phones

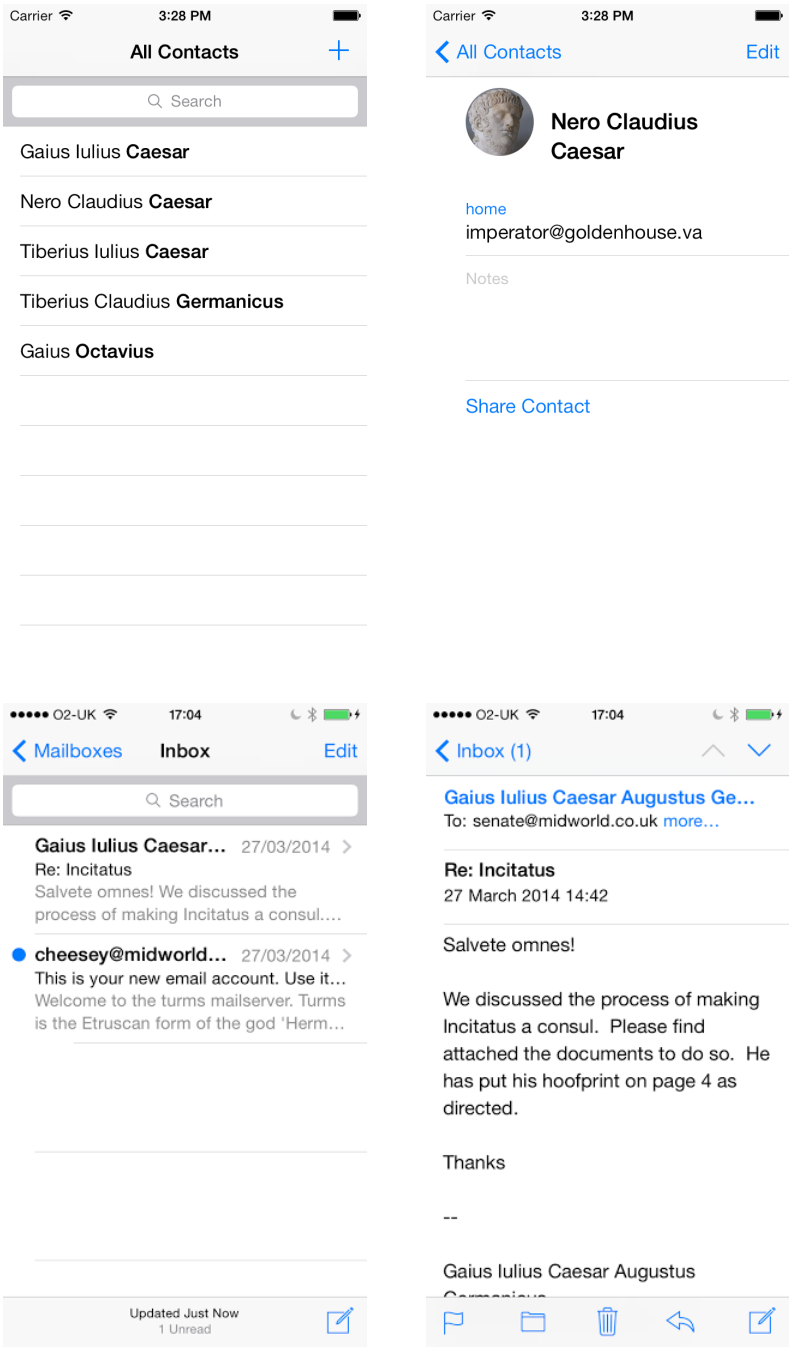


Figure 5.6: Partitive genitives on iOS phones

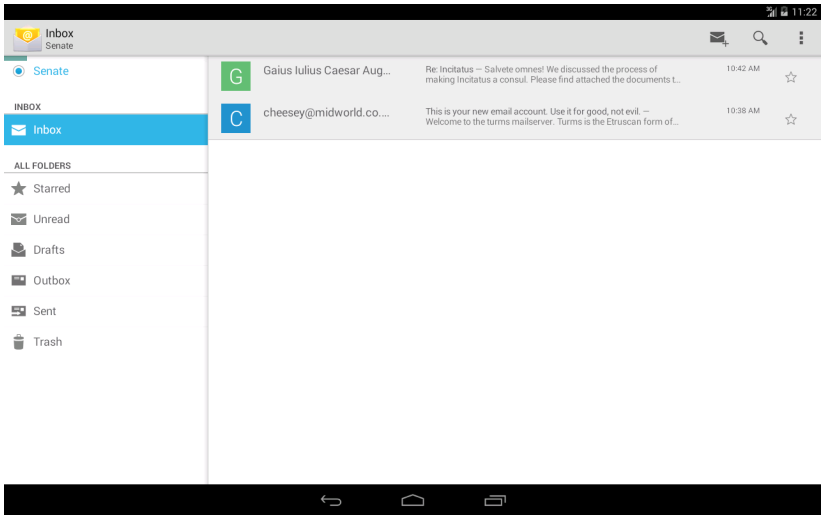


Figure 5.7: Genitive of category on Android tablets

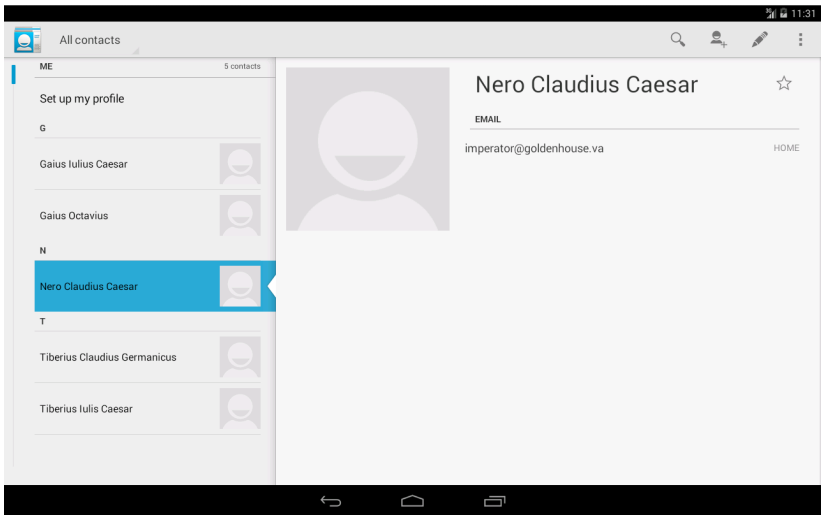


Figure 5.8: Partitive genitive on Android tablets

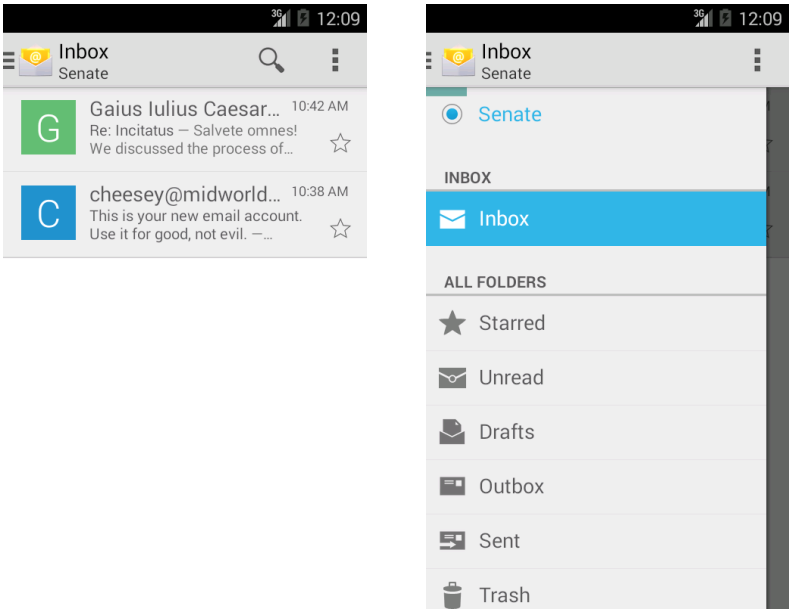


Figure 5.9: Genitive of category on Android phone

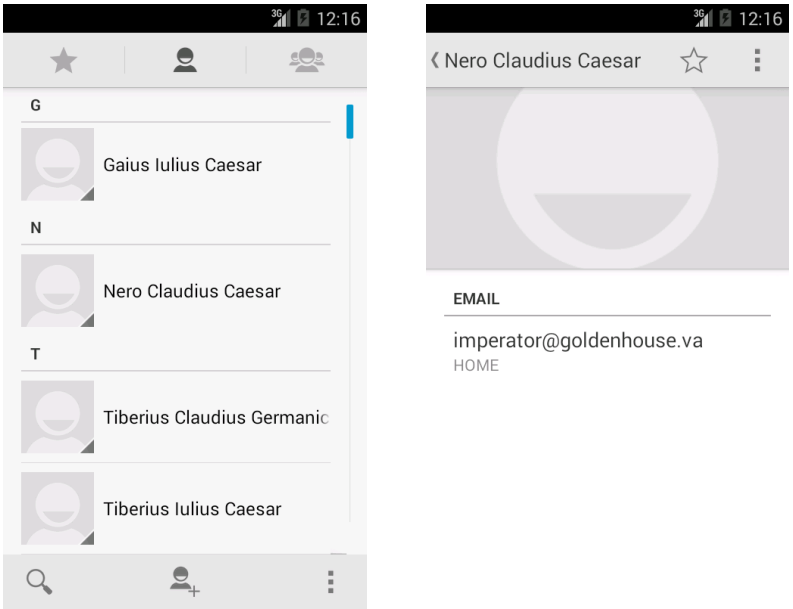


Figure 5.10: Partitive genitive on Android phone

Figure 5.11: Dative popover on iOS tablets

Figure 5.12: Dative on iOS phones

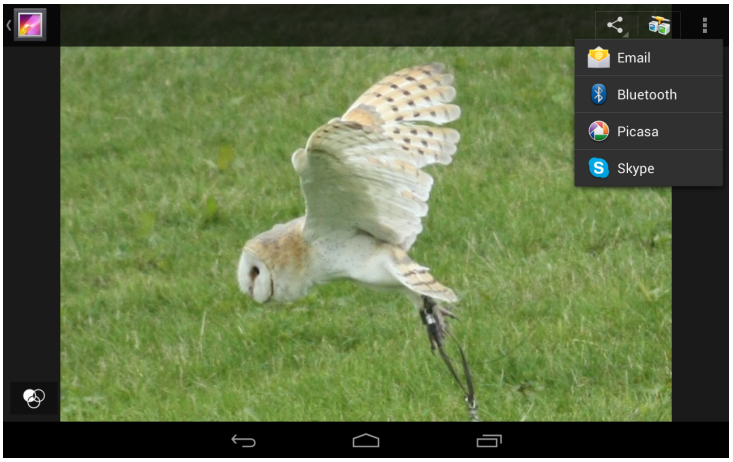


Figure 5.13: Dative on Android

On iOS phones, it is implemented as a transitory view that pops up from the bottom of the screen with the icons for the sharing destinations in it (figure 5.12). There is also a "cancel" button that dismisses the popup. In native applications this transient view is managed by an "activity view controller".

The pattern on both Android form factors is that of a drop-down menu, or a "share action provider", as outlined in chapter 3 (see figure 5.13).

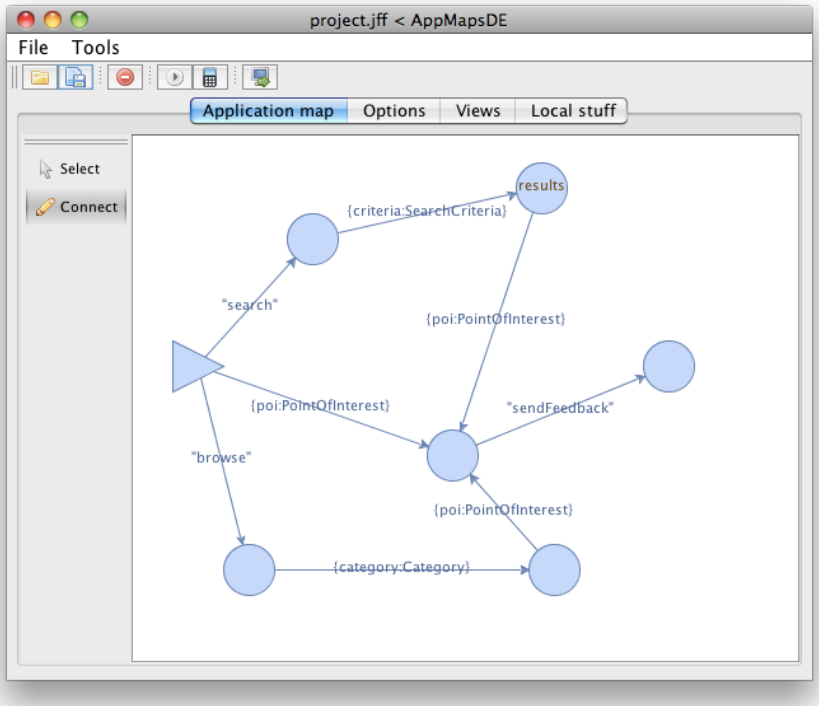


Figure 5.14: The graphical automaton editor

5.4 The dialogue notation

The case system alone, however, is not enough for a developer to specify a dialogue model. It is a categorisation, not a notation. In AppMaps, the dialogue model of an application being developed is defined in terms of a finite state automaton. This automaton is edited visually by the developer (figure 5.14).

With this model of the dialogue between user and application, it is possible to begin to do some adaptation. To describe the adaptation, it is necessary to describe both the start point and the end points of that adaptation.

The start point is the set of idioms that the developer should use to design their dialogue controller. The dialogue controllers of AppMaps applications are designed for an "ideal phone". This "ideal phone" is a simplification of both the iOS and Android phone idioms.

On an "ideal phone", genitives work as they do on iOS and in the older Android pattern (both as outlined in section 5.3.4). The user first selects the genitive, and then the other object (figure 5.15). This is the pattern that the developer should

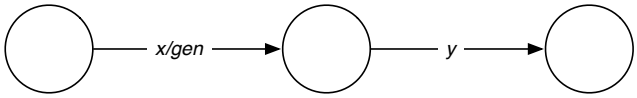


Figure 5.15: “Ideal phone” genitive pattern

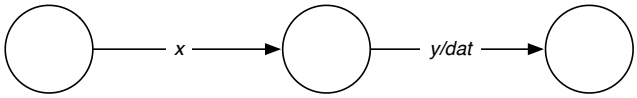


Figure 5.16: “Ideal phone” dative pattern

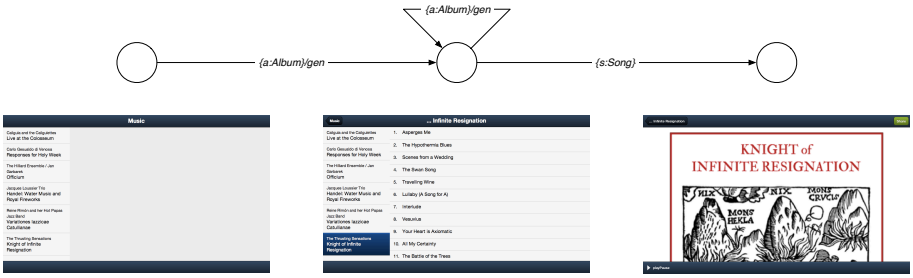


Figure 5.17: Genitive dialogue on iOS and Android tablets

use. Datives work as they do on Android: the user chooses a share button and then the dative destination (figure 5.16).

Graph grammars are used to recognise these patterns during the application build process. A graph grammar is analogous to the more usual kind of grammar (in the computer science sense of the word), but instead of replacing substrings of a string input, it replaces subgraphs of a graph input.

There is one graph grammar for each combination of software platform and form factor. Since AppMaps supports the Android and iOS platforms and the phone and tablet form factors, this means there are four distinct graph grammars in use, each of which has rules for each of the two cases. Four variants on the application thus result from the build process with four distinct concrete UIs.

On iOS tablets the genitive is rendered as a split view, as shown in section 5.3.4. To rephrase this in terms of the dialogue, the first thing that the user has to do is to select the item in the genitive. They are then able to select either the other object, or another genitive (figure 5.17).

The dative is rendered as a popover. To rephrase this in terms of the dialogue, the user first chooses the share button and then the destination to which to send the thing they are looking at (figure 5.18). Therefore, in the iOS tablet grammar, dative subgraphs are passed through unchanged.

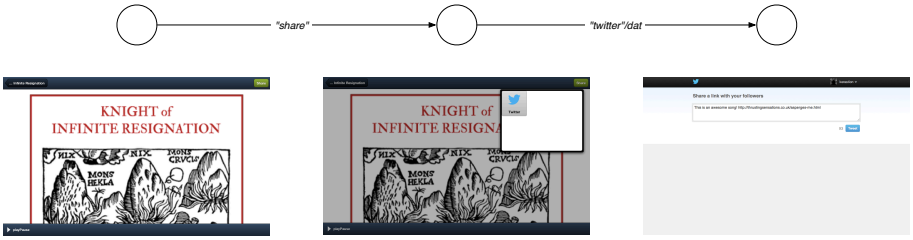


Figure 5.18: Dative dialogue on iOS and Android tablets

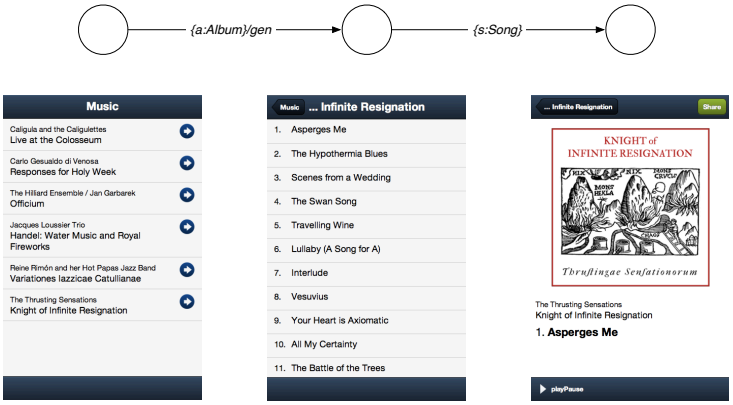


Figure 5.19: Genitive dialogue on iOS and Android phones

On Android tablets the dialogue realisations are the same as on iOS tablets (again, see section 5.3.4). Genitives again are a split view, and datives are a menu (which is, from the dialogue perspective, the same as a popover; figure 5.17).

On iOS phones genitives are implemented as a sequence of two views. In the first, the user selects the genitive; in the second, the other object (figure 5.19). Thus, genitives remain unchanged between the "ideal phone" and the real phone (compare figures 5.19 and 5.15).

Datives are implemented as a view that pops up from the bottom of the screen and has a cancel button. In dialogue terms, the user selects a "share" button of some description, then either selects the service to share to or the cancel button (figure 5.20).

On Android phones both realisations are the same as the "ideal phone". The realisation of the genitive is the same as on the iOS phone (figure 5.19); and the dative is a straightforward menu where the user picks first the "share" button and then the service they are sharing to (again, from a dialogue perspective, this is the same as a popover: see figure 5.17).

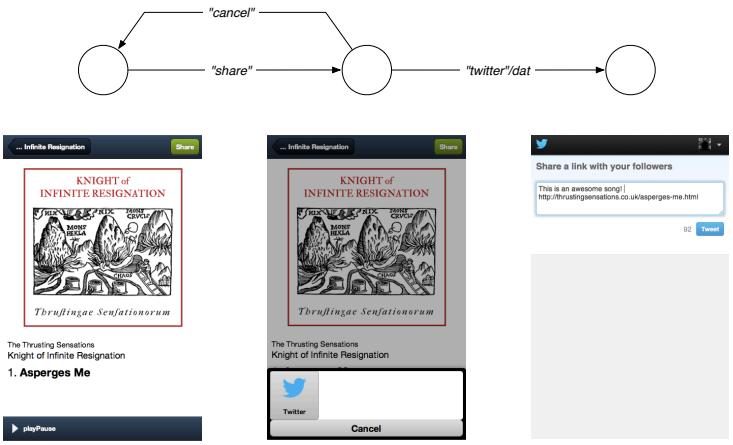


Figure 5.20: Dative dialogue on iOS phones

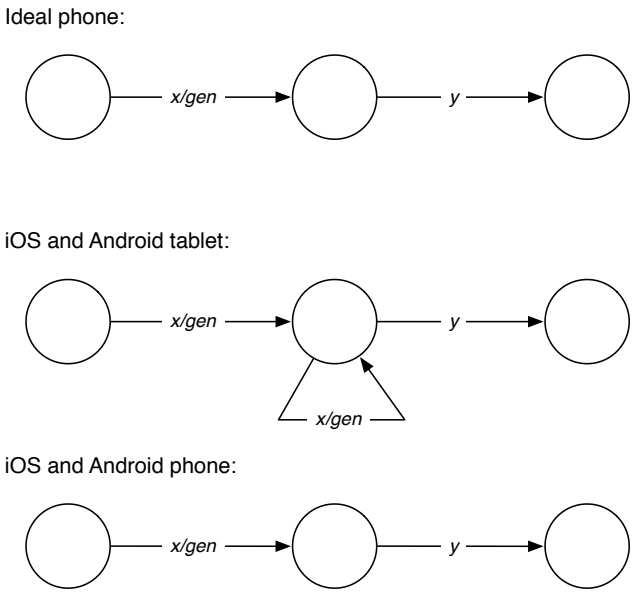


Figure 5.21: Genitive graph rules

These graph grammar rules for every platform and form factor combination supported by AppMaps are summarised in figures 5.21 and 5.22.

The specific graph grammar framework AppMaps uses is the AGG framework (Taentzer and Beyer, 1994; Taentzer, 2000, 2004). This framework was chosen after the desired grammar was designed; it was chosen because it is powerful enough to implement the above grammar rules and is easy to integrate within other software. Note that the users of AppMaps never use the AGG user interface.

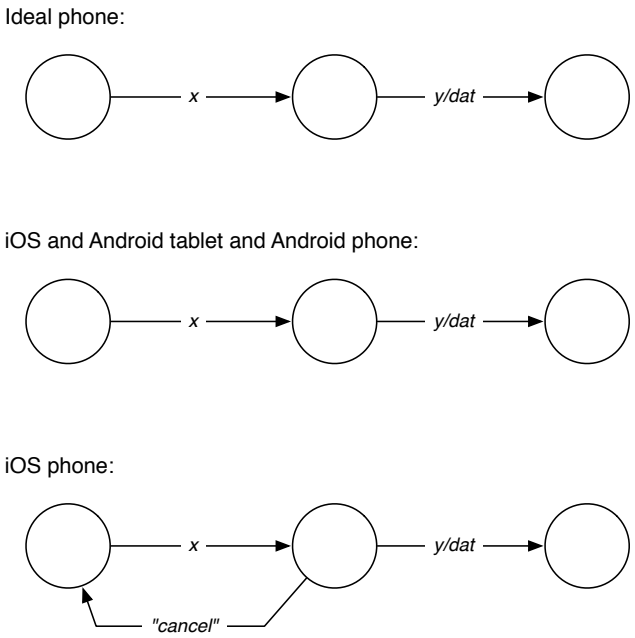


Figure 5.22: Dative graph rules

5.5 The stylesheet

The dialogue automaton does not provide enough information on its own to reconstruct the kinds of user interface pattern given in section 5.3.4. For example, consider the dative pattern for tablets as given in figure 5.17. The middle state of the three has two edges emerging from it: a genitive edge that returns to that same state, and an oblique edge that continues onwards. The screen layout calls for a split view where the left hand view, which displays the genitive, is narrower than the right hand view, which displays the oblique selections. These adaptations are on the physical layer, rather than on the dialogue layer, and so they are not the proper concern of the dialogue controller. Therefore, a mechanism for physical and logical adaptation also needs to be provided.

In AppMaps, this function is provided by a stylesheet. Each AppMaps application contains five stylesheets: one general one, which provides styling information that must remain consistent across all platforms, and one each for each supported combination of platform and form factor.

When a new AppMaps application is created, these stylesheets provide rules that implement the user interface patterns outlined above. The developer is then free either to change these rules to change the application's physical appearance or to add new rules to change the way that parts of the application appear.

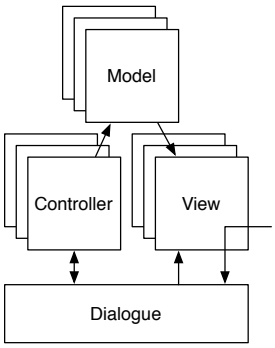


Figure 5.23: The overall structure of an MVCD architecture

5.6 Application architecture

AppMaps follows a "model-view-controller-dialogue" architecture (MVCD) based on the Sencha Touch MVC architecture. "MVCD" is a novel term: this thesis introduces it because, while individual instances of this architecture pattern already exist, there is no term for the pattern itself.

The difference between an MVCD architecture and a traditional MVC architecture is that an MVCD architecture contains a separate dialogue controller, and thus encapsulates task structure in a way that traditional MVC does not. This dialogue controller corresponds to the dialogue controller in the Arch model. It receives processed user input from view components and can either directly affect the properties of these views (such as which view is visible, or what user interface elements are enabled for user input), or can call application logic in controllers. The overall structure of an MVCD architecture is shown in figure 5.23.

These two abilities follow both from the definition of the dialogue controller and from its position on the Arch model. It must be able to co-ordinate both user actions and actions performed by the machine. In traditional MVC architectures, any operation that changes the state of a view should go through a controller; but if the dialogue controller had to go through a controller to change the state of a view, then the distinction between task structure and application logic would be blurred.

The abilities of the dialogue controller to interact both with views and with controllers directly mirror the split between the user and machine action nodes in ConcurTaskTree task models.

Apple's MVC with storyboards is an instance of the MVCD pattern; as is Sproutcore's MVC with state charts (section 2.8); as are MVC models built on top of Oracle's NetBeans flow designer (Motocoder, 2006). In the case of AppMaps, the dialogue controller is defined using the automation notation described above.

5.7 Personal reflection

In chapter 4, it was noted that it would be possible to do some reflection upon the expressiveness of the tool purely from its design. Reflections based on the design of tools are not uncommon. Jones et al. (2012) categorised and evaluated industrial tools based reflection upon on their design. Likewise, Thevenin et al. (2003) analysed tools according to the plastic user interface snowflake according to their design. A number of questions were raised in chapter 4 for this reflection to answer , and here they can be answered.

Can a case-based approach be cleanly implemented? The mapping between cases and rules from the UI guidelines is a simple one. There was no need to stretch the intent of the user interface guidelines to make each case have a simple, coherent set of realisations; nor was there any need to make the meanings of the cases implausible. In AppMaps, there is only one possible realisation of the genitive per platform. As noted above, in the general case on Android, there is more than one possible realisation. However, generally one would only use one of these realisations per application. The meaning is the same: the reasons for choosing one over the other are aesthetic or ergonomic.

The dative has two possible realisations: AppMaps again only implements one of them. In this case, the different realisations are triggered by physical properties of the component. It is worth noting in this connection that natural languages rarely have only a single realisation of a case: different words can show the same case differently (see table 3.1 for an example).

Given these circumstances, it is fair to say that there is a clean implementation of case for all the platforms involved.

How does the expressiveness of this approach compare to that of other tools? In chapter 2, the Arch model and the Plastic UI Snowflake were used to talk about the expressiveness of different tools and approaches to the design of plastic user interfaces. To compare the case-based approach used in AppMaps with other approaches, AppMaps is here classified on the plastic user interface snowflake and compared with other tools.

What software components are capable of adaptation. AppMaps allows adaptation of the physical, logical and dialogue layers of the arch model. As noted in chapter 2, industrial tools tend to only adapt at the physical and logical layers of the Arch. Thus, AppMaps is more expressive of plastic user interfaces than the industrial tools are. Note, however, that its adaptation at the dialogue level of the Arch is not as flexible as that of

the research tools: they let the developer define arbitrary transformations, where AppMaps does not.

Kinds of target. AppMaps only adapts based on the platform (in Thevenin et al. (2003)'s meaning of the word). It does not adapt based on user or environment. In this respect, it is on a par with the other development tools for the platforms it does adapt to. However, the number of platforms it adapts to is much lower than the research tools surveyed.

Development phases. AppMaps is active both at design and run-time, although all adaptation is performed at the end of design-time, as a part of the build process. At design-time, it is a forward engineering tool for creating new applications. It provides no reverse-engineering tools. At run-time, it provides a run-time component that is responsible for the life cycle of the application and its views, so it provides infrastructure rather than a toolbox. Both infrastructures and toolboxes are represented in the existing tools. It is here on a par with the expressiveness of other tools.

How the interface is implemented. The interface is implemented using JavaScript and HTML on top of WebKit. It uses the Sencha Touch 1.1 toolkit to provide physical components. It expressiveness is therefore limited compared to tools that use the native operating system components.

Which actors are in charge of adaptation. During design-time, both the computer and the user perform adaptation. At the physical and logical layers of the arch, both are involved; the user provides the stylesheet and the AppMaps tool checks it and provides platform-specific defaults. At the dialogue layer of the arch the computer alone performs adaptation. No adaptation is performed at run-time; the run-time component provides support for the adaptation decisions taken at design-time. This potentially makes the tool less expressive than the research tools surveyed in chapter 2, which allow the developer input into how the dialogue component is adapted.

When the computation of the final UI occurs. The computation of the final UI in all its details is done at run-time. The declarative specification (the stylesheets and the dialogue controller graph) is turned into concrete components when the application starts running on the mobile device. The data that populates those components, which also forms part of the final UI, is fetched by the application as needed. Data loading is under the control of the developer's own code. This puts it on a par with nearly all of both the industrial and academic tools.

Whether the final UI can migrate between devices. AppMaps provides no support for user interface migration. The only tool surveyed in chapter 2 that can provide this is X11, and to actually implement user interface migration in this environment requires both persistence and expertise on the part of the developer and user (Solomita et al., 1994). AppMaps is therefore about on a par with all the other tools in this respect.

In summary, AppMaps is generally more expressive than the industrial tools in its support for building plastic user interfaces. This does not mean that it is more expressive in general: for example, the naïvety of the history mechanism means that it would be difficult to build some applications in AppMaps that would be trivial in other tools. However, those other tools do not attempt to plasticise the user's dialogue with the machine at all, and AppMaps does, so in terms of plasticity, it is more expressive.

AppMaps is less expressive, in general, than the research tools. This follows from two things. First, AppMaps does not follow a model-driven engineering approach to the design of the interfaces it produces. It is not a model-driven engineering tool because, while it does use models and transformations between them, those transformations are not in themselves models, and because the developer has no direct access to modify and create new transformations. As noted above, model-driven engineering is not a common approach to creating user interfaces in industry, regardless of its merits, and this makes it unsuitable for the evaluation of case. Second, AppMaps targets many fewer platforms than the research tools. These limitations are not the results of the use of case in and of itself: it would likely be possible to use case in a model-driven engineering context, and it would likely be possible to use case on platforms that are AppMaps does not cater for.

What are the limits on the tool's expressiveness? The design of the AppMaps tool puts some constraints on what developers can express that are due to design decisions that are not attached to the case-based approach.

- The complexity of the dialogue structures that the notation can actually express is limited. A finite state automaton can only specify regular languages. The J2ME Flow notation escapes from these limitations by providing 'entry' and 'exit' nodes that let the application make a decision in code as to what to do, and thus making the combination of notation and extra code Turing-complete. The Apple Storyboard notation escapes this by including different kinds of dialogue controller in the storyboard, and so different parts of the storyboard can have different behaviour. AppMaps does not provide any such mechanism.

- The complexity of the dialogue structures that the notation can cleanly express is even more limited. State machines with no hierarchical constructs become very complicated very quickly.
- The dialogue controller design makes the assumption that the application can conveniently categorise the user's interaction with it into discrete selections. This makes sense for data-driven applications, such as music players or satellite navigation applications, but is less true of things like games, where the user is providing a continuous input that may directly affect objects in the application.
- The history mechanism (which implements the back button) is extremely naïve, and applications cannot opt out of it.
- The implementation of the stylesheet mechanism removes choices from the developer that might prevent them from implementing the interface they would want to: edges of the same type are always represented by the same user interface element, and styling rules cannot be applied based on the context that the user interface element finds itself in.
- Since AppMaps is a tool designed to evaluate the usefulness of case, all the plasticity it provides to the dialogue layer of the application is implemented in terms of case. Case is not a panacea for dialogue plasticity: this means, for example, that a number of potentially useful dialogue patterns are not automatically implemented at all. A very visible example of this is tabbed views. In iOS many applications have tabbed interfaces in order to group different but related functionality (see figure 5.24 for an example in the iOS App Store). AppMaps does not support this kind of pattern: if a developer should want this kind of pattern they would have to build extra code into their application to do it.

Again, none of these limitations are the fault of case in itself. The limitations of the automaton and the stylesheet come from the requirement on the tool to be familiar to developers. The limitation where all dialogue plasticity is effected by case follows from the requirement that the tool evaluate case.

5.8 Summary and conclusion

This chapter outlined the design and implementation of AppMaps, a development tool that uses the notion of case and that should be familiar to developers. It is designed for these twin purposes by beginning with a case system and then putting layers on top of it, each of which follows industrial precedents. The dialogue model that uses the case system is based on the dialogue models in use in

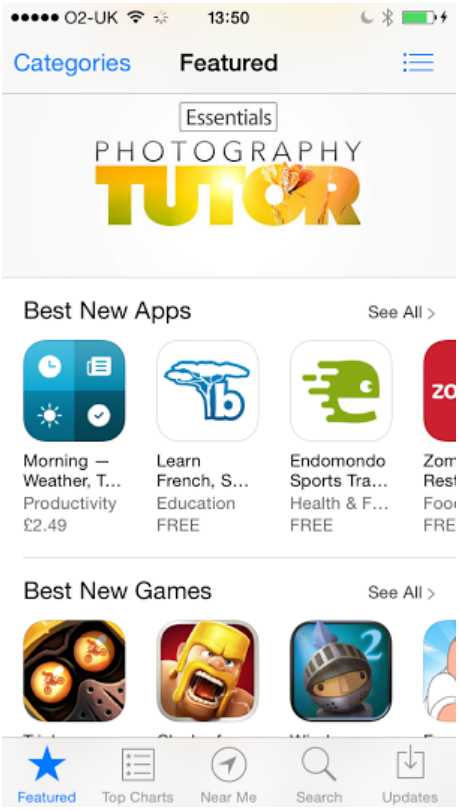


Figure 5.24: The iOS application store

other tools, especially Facebook's, Apple's and Oracle's; the stylesheet mechanism that provides some physical and logical adaptation is based on CSS and other well-known stylesheet mechanisms; and the overall software architecture belongs to a subgenre of MVC that is already widespread. These components link together to form a development tool that is capable of building applications for deployment to both tablets and phones running Android or iOS.

This chapter also performed some initial reflection upon the tool purely based on its development process. In the next two chapters this tool and the case system that is embedded inside it are put to a more rigorous evaluation first by their use in three case study applications then by their use by other developers.

Chapter 6

Development case studies

6.1 Introduction

The previous chapter outlined the design and implementation of the AppMaps tool and made some initial evaluation of it based purely on the design of the tool. This chapter continues the theme of evaluation by describing and examining the development of three case study applications using the AppMaps tool and case system. This process, as outlined in chapter 4, allows for a detailed look at the use of the case system in the practice of a developer with expertise in using the tool and the case system.

The three case study applications were free commissions. Companies who had an existing working relationship with the author were approached to submit ideas for applications. The only constraint placed on what these ideas could be was that the time available to develop the application. Prior to idea submission, the companies had not seen the AppMaps tool. There was no requirement that the application should be in any way apparently well-suited to a case-based development approach. The applications were aimed at a production release.

None of the commissions received had to be rejected due to the size of the proposed application. Therefore, of the companies approached about taking part in the study, those chosen were simply the first three that responded. Those three companies are all small, but are in very different markets. They are not related in any way other than being companies that the author has worked with before.

For each case study application, some background is given about the context into which the application fits. Any requirements for the application that affect the implementation of the user interface are then summarised, followed by the presentation and explanation of the application map for that applications. For each application, both the roles that case plays in the application itself and the roles that it played in the design process are enumerated, and the application is evaluated against the methodology outlined in chapter 4.

6.2 Goal

The goal of the work presented in this chapter is to examine how useful case is in the commercial development of plastic user interfaces at close quarters.

To this end, the applications presented in this chapter were not created purely for this thesis. Each application was commissioned by an external organisation to fill a need that that organisation had, and each application was created and developed with an aim of that application going into production on mobile devices. The applications were free commissions: the ideas for the applications came from the companies concerned and there was no restriction on the nature of the application other than that it should be possible to implement it within a reasonable amount of time. The projects were treated by the author as commercial commissions.

This allowed a great deal of access to the details of the application where case could be useful to the author's development practice and into the kinds of application where case could be useful, likewise relative to the author's professional practice.

6.3 Overview of method

In order to make sense of this chapter, it is important to keep separate the approach taken to the *research* and the approach taken to the *development of the applications*. Although the two interact, they are distinct.

The research approach is case-study based: that is, it is "an empirical enquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" (Yin, 2002, p. 13)

The development approach was one of iterative design and implementation . Initially one or two meetings were held with the representative or representatives of the company. In these meetings, the author and the representative of the company agreed on a first approximation to the final feature set and some ideas as to how the user interface should fit together. After these initial meetings, there was an iterative process where each iteration consisted of the creation of a new prototype, followed by a discussion of that prototype that set the goals for the next prototype. This discussion was sometimes a face-to-face meeting, and at other times mediated by e-mail or teleconferencing.

An application was considered ready for inclusion as a case study when both the author and the commissioning person within the company considered the user interface feature-complete. At this point, a final meeting or set of meetings was held with the representative or representatives of the company, focussing on capturing their perspective of what had gone well or badly with the development pro-

cess from their point of view, and how successful the project had been in capturing their requirements.

The audio from meetings was recorded, as was the audio from teleconferencing calls. All correspondence was also kept. The parts of the meetings that had a bearing on the case system were transcribed. These statements, along with the parts of the correspondence that had a bearing on the case system, were then classified in terms of the 3 Es. The classification was performed according to the definitions of the 3 Es presented in chapter 4. If a statement covered multiple Es, then it was classified under both. The people with whom the meetings were held were then asked to check whether the classification accurately reflected their views.

Ethics approval was granted for this study.

6.4 Cross-platform challenges

The cross-platform user interface challenges that the applications below present are quite similar. Each application (except for the Agritechnik application, as discussed below) was needed on both the Android and iOS software platforms, and on both phone and tablet. On each of these combinations of software platform and form factor they needed to follow the appropriate user interface guidelines.

In each situation, also, the requirement for a single cross-platform application rather than multiple single-platform applications was driven by two things. The first was a desire for consistency across the platforms. The people who commissioned each application all independently said that it was important that the application behaved consistently across the platforms. The second was due to the sizes of the companies: none of the companies that commissioned these applications had sufficient resources to hire developers to maintain two distinct applications.

6.5 EVENT2: Anatomy of a failure

6.5.1 Background

Between 2010 and 2013, the author was involved in developing a communication tool for conferences with a company called 1EVENT.

The tool provides two functions. The first is aimed at moderated discussions, and allows users to submit anonymous questions and comments to a screen that the moderator can see. The moderator can then choose to put any of these questions or comments on one or more big screens for discussion (as in figure 6.1). This function was used in two ways. Firstly, in debate situations where discussion was going on between a panel and the other attendees, it provided a way to keep the

question being discussed in people's vision, and thus helped keep people on topic to a certain extent. Secondly, it was used when a conference attendee wanted to voice a controversial or unfashionable point of view or question, but wished to do so anonymously. It was then up to the moderator whether this controversial content would actually be displayed. The moderator had no access to any identifying details of the person submitting the question or comment.

The second function is a polling function. The moderator can put a multiple choice question on the screen for the audience's consideration (as in figure 6.2); they can then choose one of the options, and a graph is updated on the screen (figure 6.3) to show the audience's mood. If statistically useful results are desired, the graph can be hidden until all votes are in.

Originally, the audience's interactions with the tool were all done via SMS. A number was displayed prominently, and the attendees at the conference were invited to send SMSes to it. If a poll was currently active and an incoming SMS matched one of the poll options, then it was treated as a vote and the appropriate part of the graph was updated. Otherwise, if a poll was not active, or a poll was active but the SMS didn't match any of the poll options, it was treated as an anonymous question and put into the queue for the moderator to look at.

Over time, the number of ways for users to submit their votes increased, although the core idea remained the same. MMS was introduced to let users submit images; e-mail, twitter and bridges to other instant messaging services were introduced to let users submit votes and text questions. As a part of this process, there was an investigation into building a mobile application for those users who had smartphones.

To experiment with the possibilities of mobile applications in this context, the AppMaps tool was used by the author to develop several iterations of an experimental application.

6.5.2 Requirements

Below are the requirements for the application that impacted on the implementation of the user interface. The word "server" in these requirements refers to the computer that is in charge of receiving and aggregating the messages from members of the audience. Figures are redrawn versions of those in the original specification.

- Each build of the application is specific to a conference. There is no need for any mechanism for the user to select a conference.
- As soon as the application loads, it should present the user with a tabbed screen.

Figure 6.1: The display of an anonymous comment. From the Institute of Directors Annual Debate on Jersey, 2013.

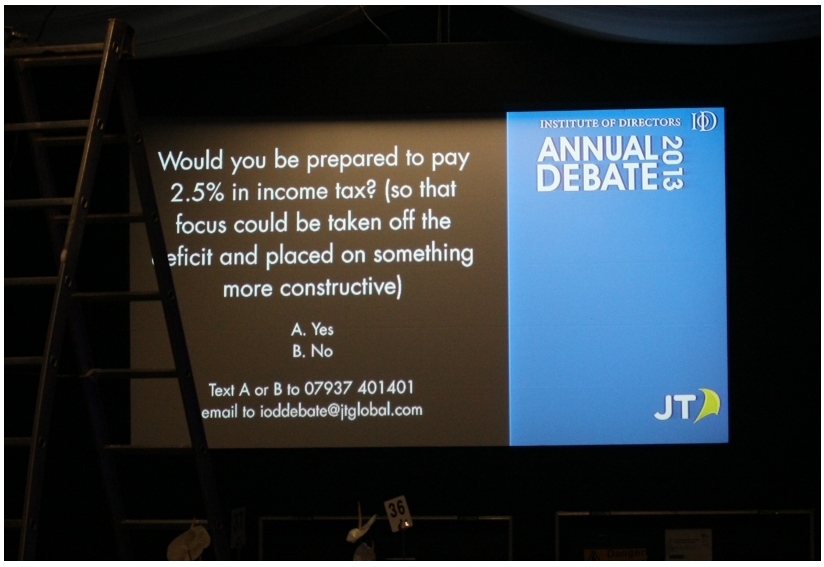


Figure 6.2: The display of a poll question. From the Institute of Directors Annual Debate on Guernsey, 2013.

Figure 6.3: The display of the graph for a poll in progress. From the Institute of Directors Annual Debate on Guernsey, 2012.

- The first tab should only be available when a poll is running. It should enumerate the poll options available and allow the user to pick one. After the user has picked one and the server has been notified, it should display a brief message indicating that the message has been passed on, and change the visual appearance of the option picked (see figure 6.4). If another option is picked then the visual indicator that the choice has been chosen should be removed from the first and placed on the second.
- The second tab should always be available. It should consist of a text field in the middle of the screen accepting up to 160 characters, and a send button in the platform-appropriate place. When the send button is pressed, one of two things happen. If the conference has only one question queue, then the message is immediately sent to that screen. If the conference has more than one, then the user is given a choice of which queue to send it to (figure 6.5).
- A third tab allows the user to replicate the main screen of the conference on their device (figure 6.6).

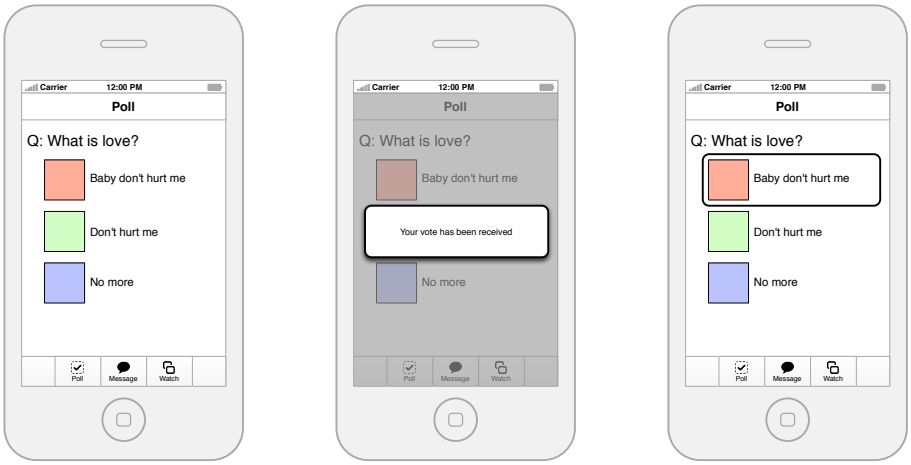


Figure 6.4: 1event poll mockup

Figure 6.5: 1event message mockup

6.5.3 The application map

The application map is given in figure 6.7. Note that the three states in the dotted box form a tab bar group: as noted in chapter 5, AppMaps does not provide automated support for this, so in this application this tab pattern was implemented separately. The dotted line is provided in this diagram for clarity and because it was in the original requirements document: it is not part of the formal dialogue automaton notation.

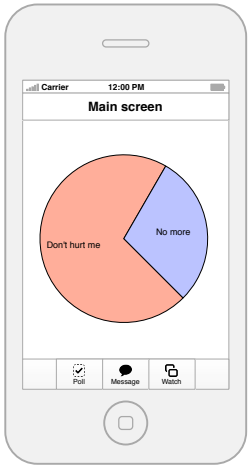


Figure 6.6: 1event main screen mockup

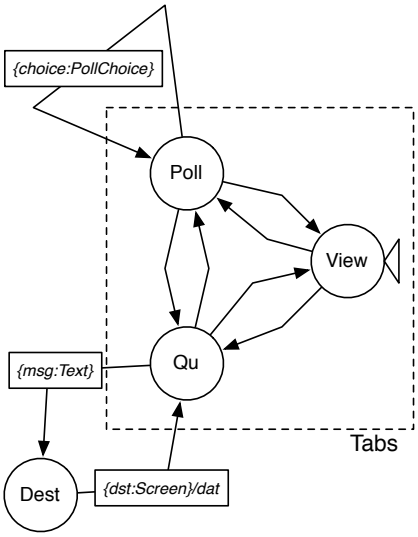


Figure 6.7: The EVENT2 application map

6.5.4 The use of case in the application

There is no use of the genitive in this application. The dative, however, is in use. It is used after the user presses the "send" button to allow them to select a screen to send their message to (figure 6.8).

6.5.5 The use of case in the design process

The features and the overall structure of the application were defined mostly by the existing product offerings and APIs; the person who was responsible for signing-off the design of the application wanted the dialogue design of the application to follow the existing dialogue flows of the SMS-based application closely. In the initial design, therefore, case played little part.

The initial prototype of the application did not have the multi-screen function. At that time, in fact, the whole product had no such function: there was only one screen per deployment. Support was subsequently added to the entire system for multiple screens after the design of a prototype application that had this support that was built using AppMaps to demonstrate the use of dative edges. This feature was subsequently removed: during the one and a half years in which it had existed not a single customer had asked for it or used it. However, AppMaps did make the addition of this feature to at least the user interface of the application trivial, and without that the entire experiment could not have taken place.

6.5.6 Evaluation

6.5.6.1 Effectiveness

Two people inside InstantVue had access to the application while it was in development and thus could comment on the quality of the application. One was the commissioning person; one was the person responsible for marketing the product.

Since only the "multiple screens" function was plasticised with the aid of case, these people's reactions will only be given as far as they touch on this function.

The commissioning person did not even initially notice the plasticity in this component of the interface. He intuitively used it correctly, but did not consciously notice that the presentation was different between the phone and tablet builds. When it was pointed out to him that this adaptation was taking place, he commented that "it works exactly the same as all the other apps on this thing, so I didn't really notice". He was asked whether this meant that he considered it to be consistent with the other applications on the system, he said "it looks a little different from the vendor apps on here, but I've seen much odder third-party apps. I've seen loads that look like this". By "odder", he said that he meant "not things that try to be different or design-y, but just badly-built attempts to look like

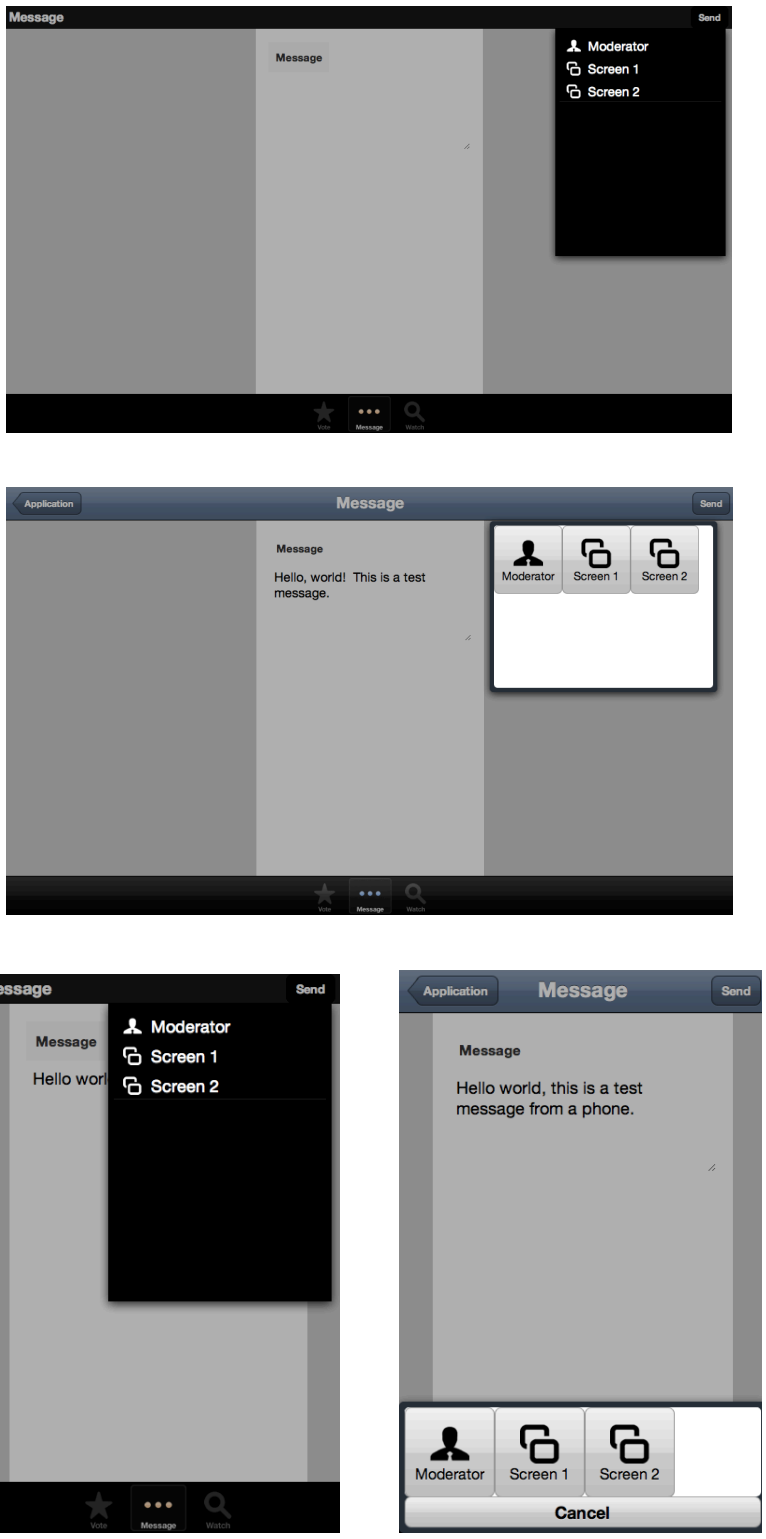


Figure 6.8: Datives in EVENT2

normal apps. This is fine, completely within what I'd consider 'normal'." When asked for aesthetic critique, he replied that "there isn't much to like or dislike aesthetically, really, other than I dislike the Android interface in general, it's ... awful, really naff. But the app is just like all the other apps. It looks nicer on iOS, but so does everything else."

The person responsible for marketing the project, by contrast, noticed the plasticity immediately. It was, as noted above, his insistence on using this that lead to the multiple screens feature of the product: "I've tried this on Android and iOS and the 'send' menu looks right on each but I don't think we're using it for the right thing. I was thinking, though, about venues that have different screens... Can we sell something where different screens can do different things?" When asked whether he considered it consistent with other applications available, he said "well, it feels right. It looks a bit weird, but then all the apps built with HTML5 do—this is built with HTML, isn't it? It looks like it," When asked to clarify what he meant by "weird" he said "The typography is wrong, the fonts look weird, the icons look slightly odd, but that's what HTML5 apps are like."

In chapter 5, the dative on iOS is implemented in terms of the look and feel of iOS's "Activity Controllers". The question of conformance to the user interface guidelines thus hinges on whether this is being used appropriately in this application.

An Activity Controller may contain both application-specific and system-supplied activities. In the case of this application, all the destinations are application-specific. The destinations are suitable for activities . The Activity Controller should display in a pop-over on the iPad and an transient view (an "action sheet" in Apple parlance) on the iPhone: and the dative appears correctly in this application (figure 6.8 above). Individual dative elements are displayed as buttons with an icon above the title.

However, most of the requirements imposed on activities are to do with physical adaptation, specifically the appearance of the icon. The icon that represents the activity should be monochrome for an application-specific activity, with appropriate alpha transparency, and must not include a drop shadow (Apple, Inc., 2012a, p. 152). These are not issues that case can sensibly attempt to address, at least in the way that it is implemented in AppMaps. In addition, an activity view controller must be summoned by a button that has a system-appropriate look and icon: this is not something that case does in the context of the AppMaps system, but it is conceivable that a more complex styling system in conjunction with case could provide this.

On Android, the dative is implemented in terms of the look and feel of Android's "action provider" that is used (via a specialised subclass) for sharing destinations (see chapter 5). On both phone and tablet, this shows up as a menu. The

dative appears correctly on both of these platforms. Individual actions within the action provider are displayed as items with the icon to the left of their name.

6.5.6.2 Efficiency

In terms of the development of the application itself, in isolation from the rest of the system, case provided a tiny efficiency benefit. The use of a pre-defined dative category that very obviously matched the user's "multiple screens" destination choice, and the corresponding default styling rules, was perceived by both the author and the person representing InstantVue as saving time.

In addition, fewer software components needed to be modified to support plasticity than would have been without case. Multiple dialogue controllers would have been needed for the multiple platforms (since the dative on iOS phones requires a different dialogue flow to that on iOS tablets and on Android). Likewise, different versions of the share palette would have been needed at least for iOS tablet, iOS phone and Android. The number of software components would have thus been linear in the number of platforms and form factors supported.

6.5.6.3 Expressiveness

Most of the plasticity in the EVENT2 application was not provided by case or by the AppMaps tool at all. Nearly all of the plasticity was done at the physical layer of the arch. The "preview" of the main screen that could be displayed on the mobile display needed to change size and aspect ratio based on the size and orientation of the screen. This was effected through the mechanisms already in the product that were also used to adapt the display to different sizes and aspect ratios of projection screens. The forms that allowed the user to enter an anonymous question or respond to a poll both needed to remain centred on the screen, and needed their text sizes adapting for different platforms. This was done through standard mechanisms built into Sencha Touch and Webkit and through the stylesheet mechanism in AppMaps. Case had no part in it.

6.6 Agritechnik: Anatomy of a partial success

6.6.1 Background

The second case study follows an application developed for a company called Agritechnik. Agritechnik are a small company who have a focus on servicing and repairing agricultural equipment. Much of the equipment they work on is very specialised. This means that the number of companies able to service this equipment is quite low. Therefore, this small company has a catchment area of most of the south of England and has customers all over northern Europe. Much of the

equipment they work on is also very large, and must be serviced on-site. Because of these two circumstances, the people who do the actual physical repairs do a great deal of travelling. This is both expensive and time-consuming. Having the ability to do initial diagnostics before sending someone out to repair the equipment was consequently perceived as beneficial for the company, for the company's employees, and for the customers.

Many of Agritechnik's customers have smartphones running iOS or Android. Because of this, Agritechnik wanted an application that would let customers send a series of photographs of the equipment needing servicing, including things like the vehicle identification number plate and any part of the machinery that is obviously worn and damaged, along with some textual details. Having these details in advance would make it easier to plan what needs to be done before the expert on the equipment actually arrives, and would also allow the company to provide more accurate quotations in advance.

Agritechnik also sell and hire out agricultural equipment. As an additional feature, they wanted their mobile app to display the items that they had for sale or hire in a browsable list.

6.6.2 Requirements

Here follow the requirements for the Agritechnik application that impacted on the user interface. Again, figures are redrawn versions of those in the original requirements document.

- When the user loads the application they should be given either the option to browse items for sale or for hire, or to submit a new request for quotation or discussion (figure 6.9).
- When the user chooses to browse the equipment for sale or hire, they should be presented with a list of currently available items. When they select a piece of equipment from this list, they should be presented with its details and a button that they can use to contact Agritechnik about the item. They should also be able to share adverts on social media (figure 6.10).
- When submitting a request, the user will be guided through a multi-step process.
  - First, the application will ask them to enter their contact details. If the user has previously given their details, then the form should remember them. When the form is dismissed, the details will be saved on the phone (figure 6.11a).

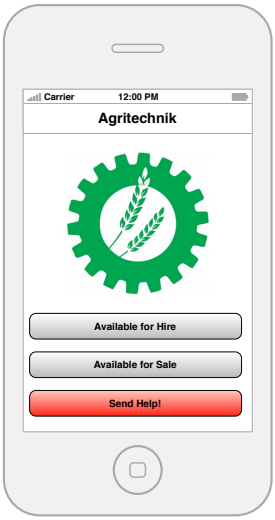


Figure 6.9: Agritechnik application front page

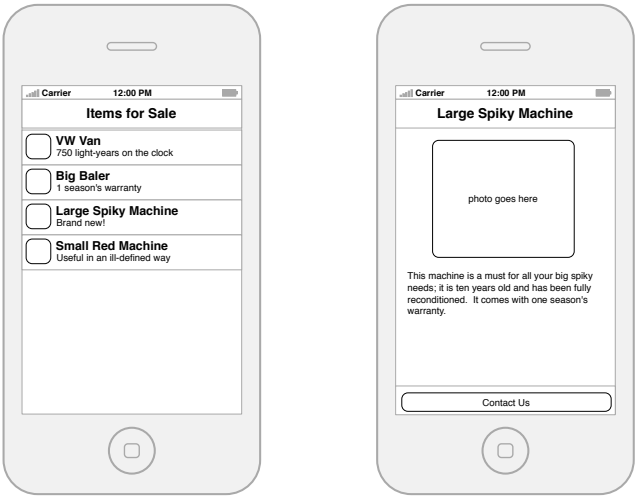


Figure 6.10: Agritechnik advertisement pages

- Secondly, the application will prompt the user to take a photograph of the equipment in question's Vehicle Identification Number (VIN) plate (as in figure 6.12). The VIN plate contains information such as the date of manufacture of the vehicle and what kind of vehicle it is. If Agritechnik have serviced it before, it provides enough information to let them consult the service history of the vehicle. If the equipment has no VIN plate, then there should be an obvious button to skip this step. An example VIN plate should be shown on this page (figure 6.11b).
- After this the user should be able to take as many further photographs as they need to in order to show the problem (figure 6.11c). After each, they should be prompted as to whether they wish to take another (figure 6.11d).
- When the user has taken enough photographs to illustrate what needs doing, they should be able to enter a textual description or message, and then select the department of the company to which the request should be sent.

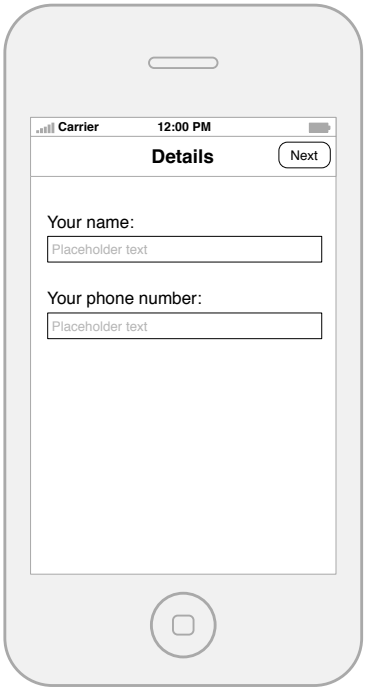
6.6.3 The application map

The application map for the Agritechnik application is shown in figure 6.13. The area below the dotted line comprises the advertisement browser: the area above comprises the 'remote help' part of the application. As in the EVENT2 application map, the dotted line is included in the figure for clarity and because it was in the original design documents: it is not a part of the formal notation.

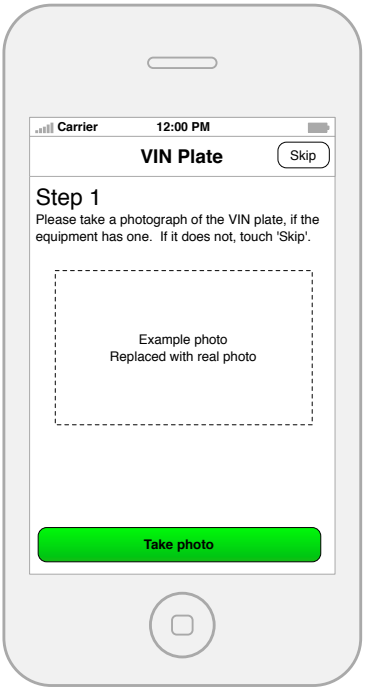
6.6.4 How are cases used?

The genitive is used in the advertisement display section of the application, to mark a relationship between the advertisement and its details. Note that it is not marking a relationship here between a category and elements of that category., but a more general parent-child relationship between the advertisement and its contents.

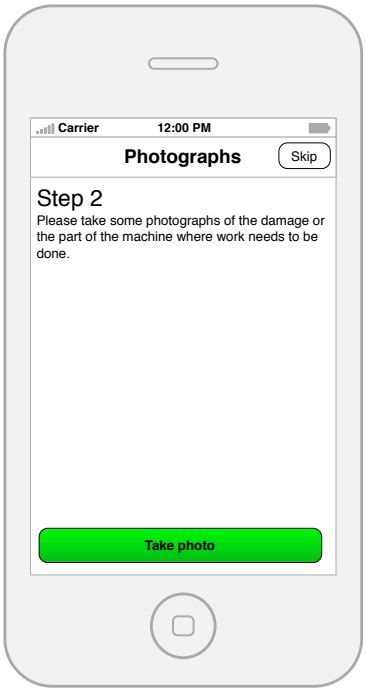
The subgraph involving the genitive is of the form shown in figure 6.14. The first state allows the user to select an advertisement. The view that is used to allow the user to choose an advert is a list that shows both a thumbnail photograph (if one exists) and the title of the advertisement. The second state shows the contents of that advertisement using a details view (see chapter 5 for definition) and provides a share button. On a tablet platform, the genitive rules given in chapter 5 transform this into a subgraph of the form shown in figure 6.15.



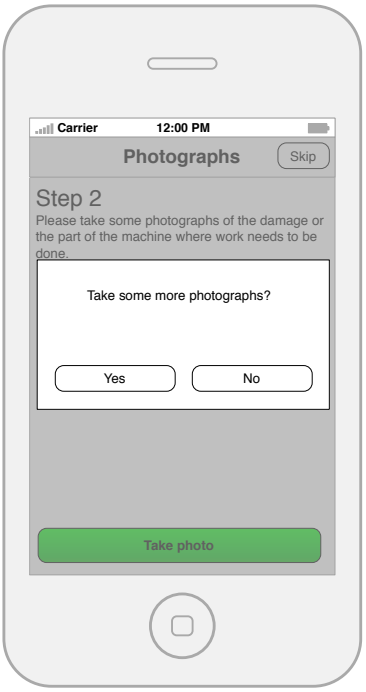
(a) Details page



(b) VIN Plate page



(c) Taking another photo



(d) Confirming further photos

Figure 6.11: Agritechnik diagnostic screens

Figure 6.12: A VIN plate from a vehicle

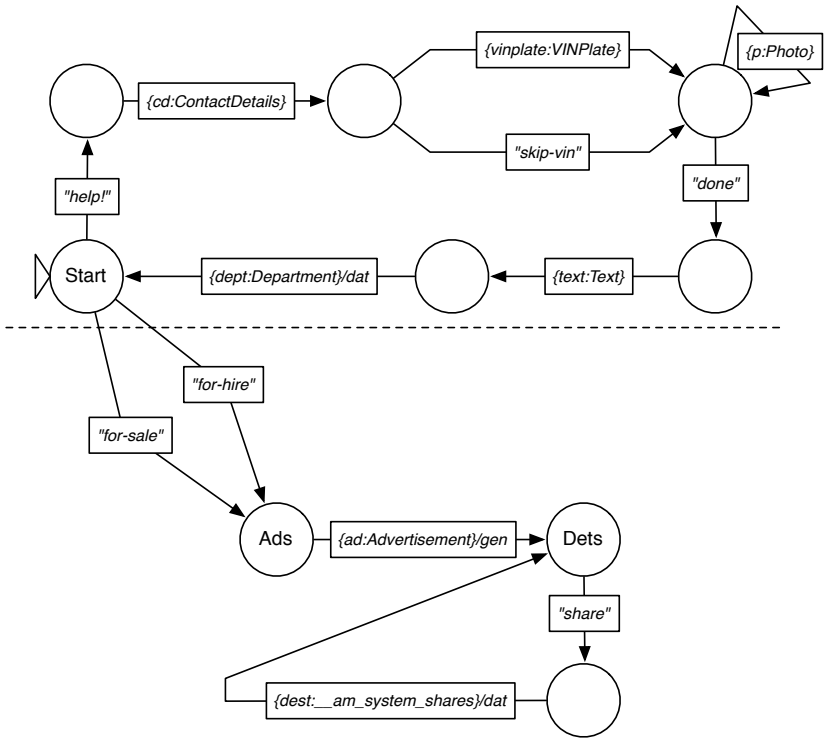


Figure 6.13: The application map for the Agritechnik application

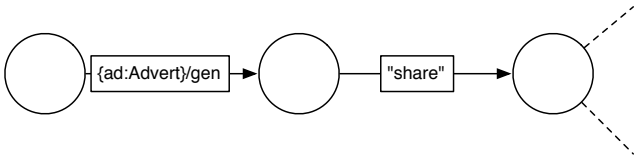


Figure 6.14: Genitive subgraphs in Agritechnik application

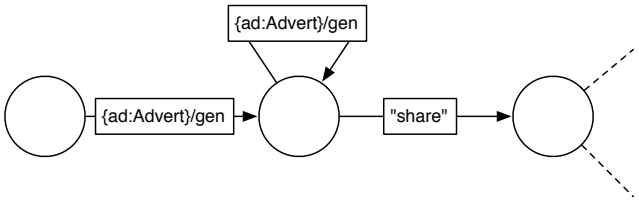


Figure 6.15: Adapted Agritechnik genitive subgraph for tablets

This transformation, along with the default stylesheets, results in iOS and Android Tablet versions both using a split view where the list of advertisements is on the left hand side of the screen. The user is able to pick advertisements from the list to see in the right hand, detail pane (figure 6.16). This is an appropriate use of the split view on both platforms.

The dative is used in both the advertisement display section of the application and the help request submission section of the application. In the former, it is used to let people either email the advertisement to themselves or to share it on social media (figure 6.17); in the latter, it is used to let users choose the department of the company to which they are sending their request (figure 6.18).

It is worth noting here that the two uses of the dative are connected only by their meaning (both being about destinations) and by their eventual realisation in the physical layer of the user interface. The actual implementations of the actions behind them are entirely different. The first uses the application programming interfaces of the social networks in question, and simply sends the contents of the advertisement and its URL to them; the second constructs an email internally, then passes it to the operating system to be sent.

6.6.5 How were cases used during the design process?

Case was directly used as a tool in two situations during the design of the application. The first situation involved genitives: there were two potential ways of structuring the advertisement browsing system. The first way involved focusing on

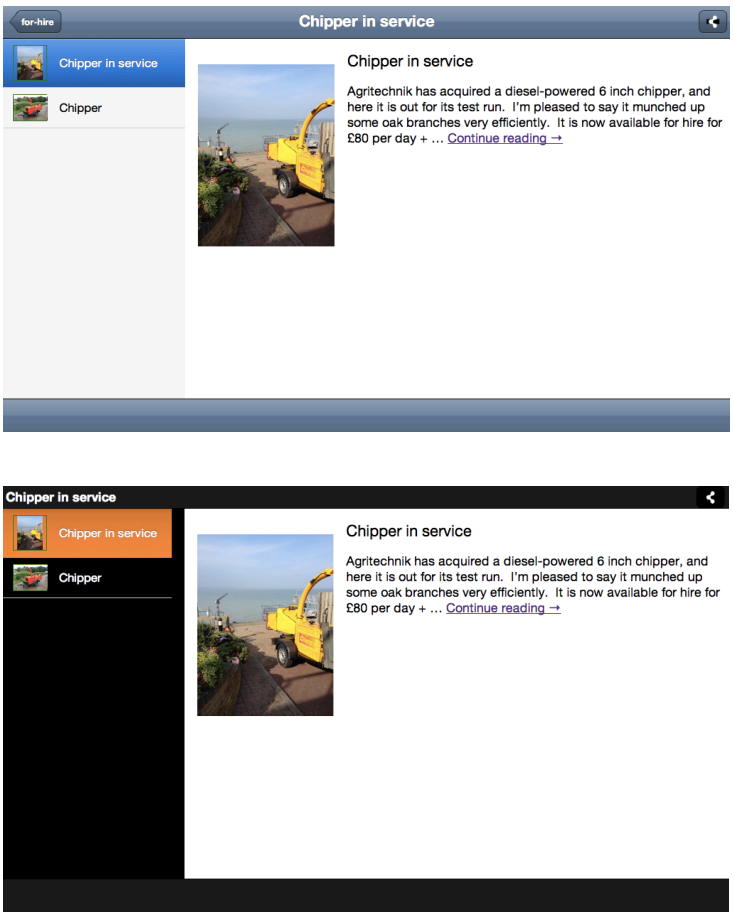


Figure 6.16: Split view genitives in the Agritechnik application

the list of advertisements: the 'for sale' and 'for hire' were viewed as categories of advertisement, and choosing an advertisement displayed that advertisement full-screen (figure 6.19). The second way treated 'for sale' and 'for hire' as separate branches on the initial screen and focused on the advertisement itself; the list of advertisements was marked as genitive as in the final application map above (figure 6.20). A third way, where both the list of kinds of advertisement and the list of advertisements themselves were marked as genitive, was considered, and the resulting interface works analogously to the mailbox view in Apple's Mail application. This was probably the most correct according to the Apple human interface guidelines. However, it was found to be unworkable due to technical limitations in the AppMaps dialogue controller.

The second situation involved an unexpected request to migrate the application to the Windows platform (figure 6.21). There were two major possibilities for the organisation of the user interface of the application based on different design

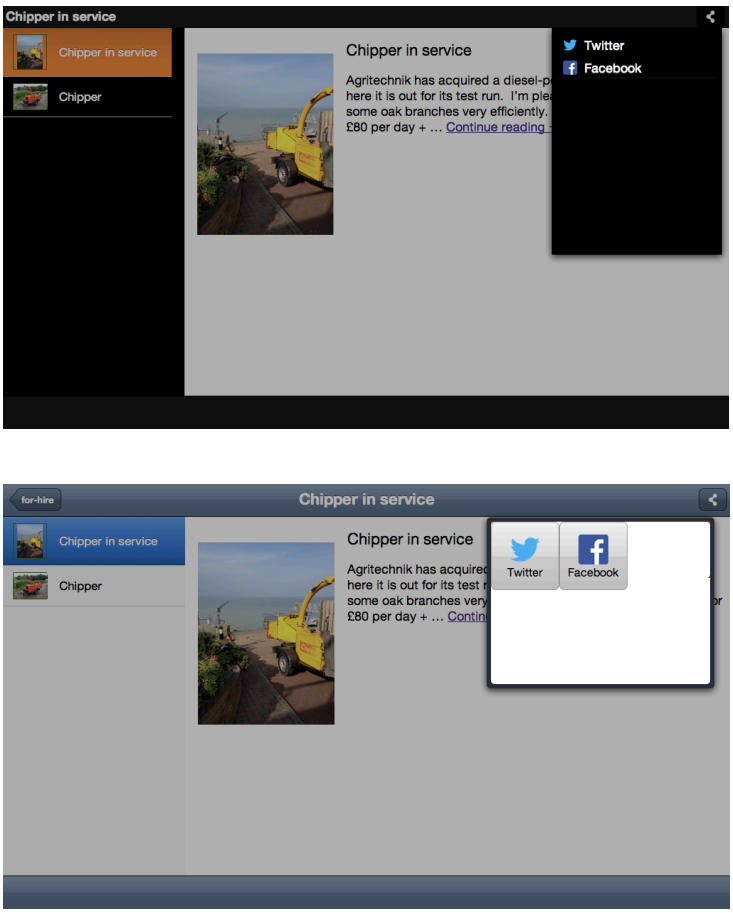


Figure 6.17: Datives in advertisements

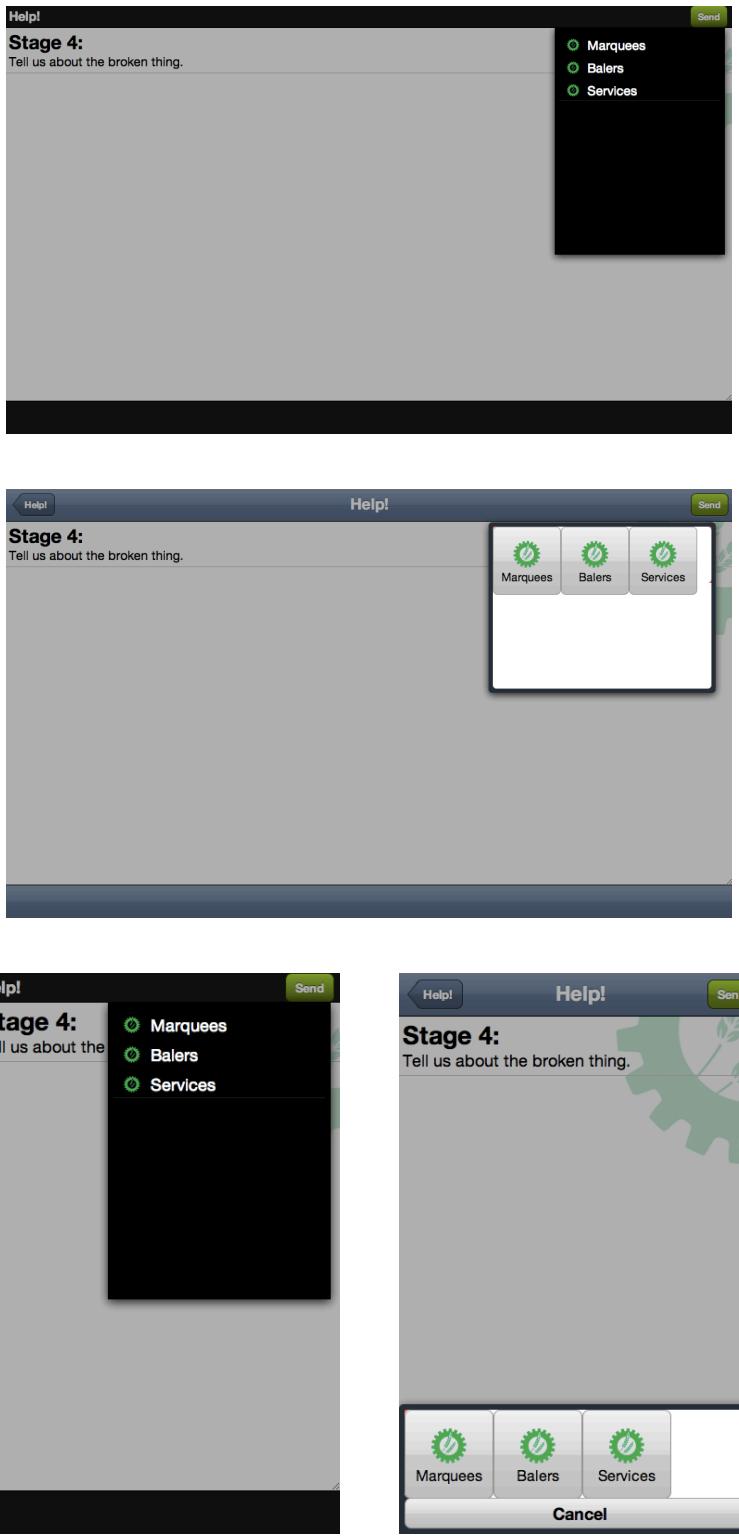


Figure 6.18: Datives in diagnostics

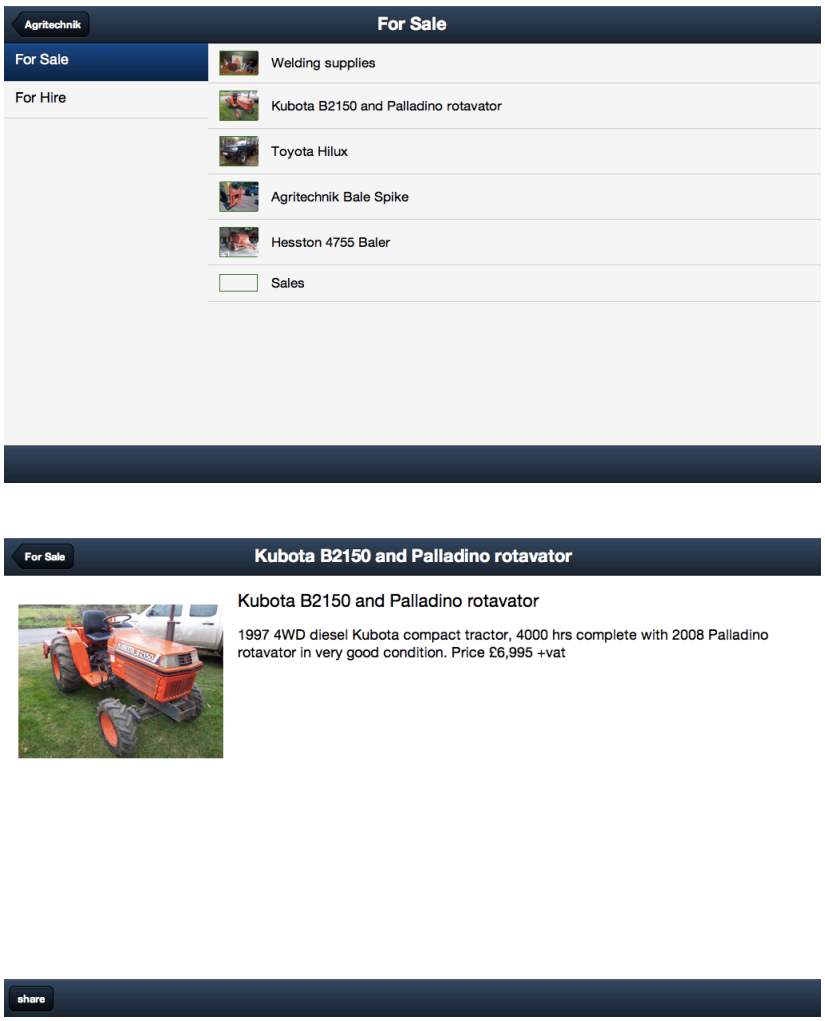


Figure 6.19: Advertisement dialogue: option 1

patters from the Microsoft Human Interface Guidelines. In the case of this application, these could be usefully expressed in terms of case, and by generating two graph grammars and two sets of default stylesheets, the commissioning user was able to choose between the two options based on concrete, usable versions of the application, rather than on wireframe prototypes.

6.6.6 Evaluation

6.6.6.1 Effectiveness

The application was developed in conjunction with a single person inside Agritechnik. This commissioning user was asked to give their feedback on the application,

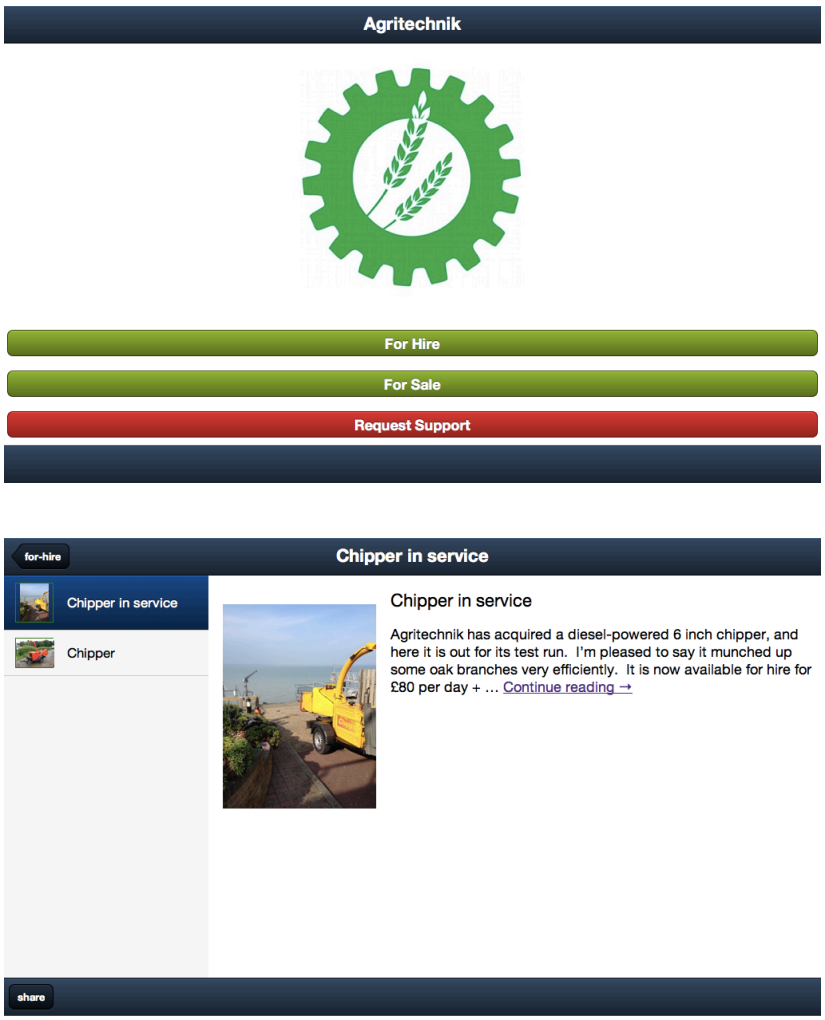


Figure 6.20: Advertisement dialogue: option 2

with specific emphasis on the ways that it moved between platforms and form factors.

Case is used in both the advertisement browsing sections of the application and at the end of the "request help" section of the application. Therefore, only the reactions of the commissioning person to those specific parts of the application will be summarised below.

They perceived the advertisement browsing section as "look[ing] at home as far as I can see. I mean, I don't use Android, so... but it certainly looks OK on my iPad as well [as the on the iPhone]. The other version you sent, with the 'for hire' and 'for sale' at the side were clumsy, even though the photos were bigger on the adverts." When asked for more details on what they meant by 'clumsy' they said

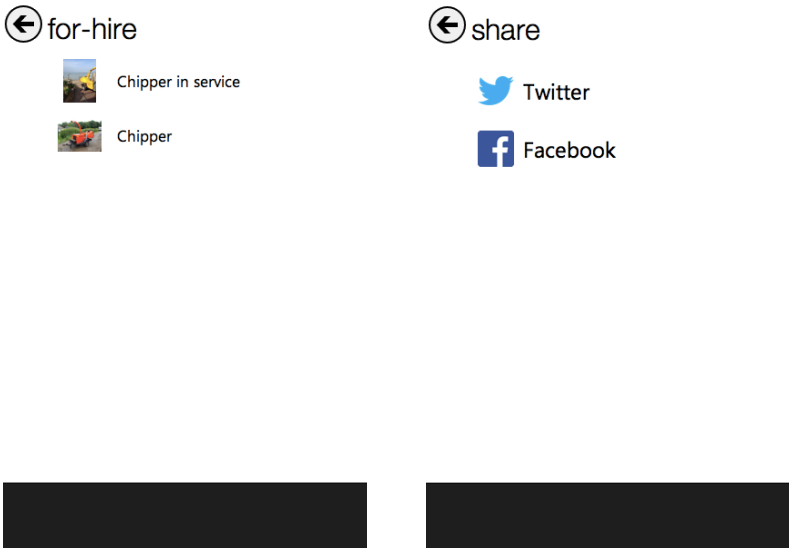


Figure 6.21: Agritechnik application on Windows Phone

"... it felt right having the adverts on the left and the contents on the right on the tablet, it works like Mail and Skype and everything, with a set of things on the left and you're looking at stuff inside those things on the right. For sale and for hire are two different things and you're mixing them up." Later, on being asked about the Windows Modern UI version of the application, they said: "... it [the tablet version]'s quite similar to the phone UI on Windows, isn't it... it obviously means the same thing as the iPad and iPhone version of the app, but it looks and feels like Windows. There's not much there to have an opinion on, it works, doesn't it?"

With regards the dative for sending to departments of the company at the end of the "request help" function, they said "Yes, fine. It looks like everything else on the iPad and iPhone and as far as I can see on Android and Windows too. This came out a lot better than I was expecting, really. I was expecting it to just send at once as soon as you press the button but it pops up the email for them to fix all their typos before they send it. Other apps seem to do that, too, for twitter and stuff, they all pop up a box with settings or a message or something, and this does too, which is OK."

This comment is important because it points out a convention that AppMaps as it currently stands cannot adapt to different platforms. Destinations, in all the platforms in use, are often followed by a step where the user confirms the details of what they're sending and to where they are sending it. This is simply because different kinds of destination require different kinds of clarification: the extra details the user would provide to send data to a social network (such as a

textual message and possibly a more specific destination user) are clearly radically different from the kinds of extra details the user would provide to send data to a printer (such as paper size and orientation). Sometimes this details prompt is managed by the underlying operating system (such as a popup requesting extra details for a print job). Sometimes, this details prompt is managed by a piece of software that is neither an integral part of the application nor part of the operating system (such as the Twitter and Facebook JavaScript components or an external e-mail client). For destinations that are internal to the application, however, the application needs to manage this details prompt (if needed) itself.

Because of the variability in the way that sharing destinations are implemented, sometimes this can be represented as an edge in the application map, and sometimes it cannot. While this is an issue with the AppMaps implementation of case rather than with case in the abstract, it is good evidence that the developer would need to keep in mind the user interface conventions for the platforms for which they are developing even with a more expressive dialogue controller.

To evaluate the parts of the interface that are affected by case against the platforms' human interface guidelines, three items need to be considered on each platform. The first two are the realisations of the genitive and dative in the advertisement browsing section, and the third is the realisation of the dative in the communication section of the application.

On the iPhone, the genitive is a full-screen view. If it is a table view (Apple, Inc., 2012a, p. 168) then the items in that table view are given disclosure indicators. In this application, this was not correct: disclosure indicators should only be applied to items in a table view if, when selected, they display another table view. In this application, when the items were selected, they displayed content instead (compare figures 5.5 and 5.6 in chapter 5). This is an issue at the intersection between the physical and logical layers of the arch. Since case has input into adaptation at the logical layer, it would seem that this is a problem that the combination of case and stylesheet should be able to overcome. However, in the AppMaps implementation of case, this is impossible, for reasons outlined in chapter 5.

Neither of these problems are inherent to the case system. While this application provides no evidence that case can be used as part of an approach to this kind of context-specific logical adaptation, nor does it provide evidence that case cannot be used here.

By contrast, on the iPad the genitive is indicated using a "split view" (p. 166), and uses it in a textbook manner (see p. 167). The selection on the left corresponds to an object that the user is to view the details of on the right. Apple note that this is such a common use of the split view that the popular terminology for the

left pane is "the *master pane* and the right pane is often called the *detail pane* ... this relationship is not enforced in code."

The Android human interface guidelines are more vague about how this kind of meaning ought to be conveyed. For devices with small screens, generally only one "pane" should be visible at once. Genitives in AppMaps on small-screen android devices are displayed as two screens, one after the other, and this meets this requirement. No disclosure indicators are permitted in lists (sections "Lists" and "Pure Android" in Google, Inc., 2013a), and none is added.

By contrast, the guidelines suggest that user interfaces for larger screens should use "multi-pane layouts" (Google, Inc., 2013a, "Devices and Displays"). This is similar to Apple's "split view" but allows for more flexibility as to how the content is presented. Unlike Apple's "split view", where the leftmost panel is of fixed width, multi-pane layouts can have variable widths. The use of this pattern for this application, where the left pane shows the list of objects and the right pane shows the details of the selected object, is non-controversial (s. "Multi-pane views").

The uses of the dative in the advertisement browser and in the communication section of the application (figures 6.17 and 6.18 above) are very similar to the use of the dative in the EVENT2 application above. Again, both appear correctly in a pop-over on the iPad and in a action sheet on the iPhone, and both use an "action provider"-like interface on Android. Again, the failures in plasticity are due to requirements at the physical layer, specifically those involving icons (the same requirements apply as in section 6.5.6.1 above).

6.6.6.2 Efficiency

In the Agritechnik application, only two software components effect plasticity: the stylesheet and the dialogue component. None of the views, controllers, or attendant classes check anything about the platform or form factor they are running on. The stylesheet only contains one rule that is platform or form factor specific: on Windows tablets, genitive edges that correspond to advertisement lists are passed a style option to tell them to display more information about each advertisement. Since this rule is defined in terms of the cases on edges, it is reasonable to assert that all the plasticity in the application is achieved by means of the use of case.

If the same end result were to be attempted using the facilities built directly into Sencha Touch (which are similar to those built into the other tools discussed in chapter 2) then either four or five components would have had to implement plasticity:

- The two views that implement the advertisement browsing screens. Each of those would need to be aware of the platform and form factor running

the application, so that it could arrange its component parts correctly on the screen. If these two views were implemented as instances of the same view class, then they would account for one software component instead of two. They also need to ensure that they display the "share" destinations in the correct place on the screen.

- The two views that implement the "destination" selections, the one for sharing advertisements and the other for choosing which department will be the destination of the request for help. Each of these needs to adapt itself to display correctly on the platform and form factor in which it finds itself.
- The last screen in the 'request help' sequence of screens needs to make sure it displays its list of destinations in the correct place on the screen.

In this scenario the number of software components that must be modified to effect plasticity scales linearly in the number of views that change form based on form factor or platform.

A fairer comparison is one with a hypothetical development system that possesses the dialogue component and stylesheet but does not use case. In this case, each pair of platform and form factor would require its own dialogue map. This means that the number of software components that would be required to effect plasticity would scale linearly in the number of platforms and form factors involved.

The result suggested above, where the adapting dialogue controller and stylesheet make the number of software components required to effect plasticity constant is not, of course, unique to case. However, if case is *expressive* enough (in the sense of chapter 4) to provide the adaptation needed to keep this number of software components involved near-constant then the efficiency gain can properly be said to come from case where case is used.

As laid out in chapter 4, the author's subjective impressions about the efficiency of the development process can also properly be considered here. From the Agritechnik application, two dominant subjective impressions emerged about the development process compared to the author's previous development projects.

First, there was the perception that much less boilerplate code needed to be written during the development process, especially styling and dialog structure code. Much of the credit for this must go to the dialogue controller and stylesheet mechanism, rather than to the presence of case: certainly, compared to writing applications in unadorned Sencha Touch, having the dialogue controller removes the need to write code for back buttons and for sequencing, which are omnipresent. However, even compared to writing applications of similar complexity using the recent versions of Apple's development tools that do include a dialogue controller,

boilerplate seemed reduced. Notably, in Apple's scheme, the developer has to maintain two dialogue controllers for the two different form factors that iOS supports (see chapter 2). Likewise, even in the Apple system with the dialogue controller, duplicate and boilerplate code for user interface elements that fulfilled functions that would be under the same case in the AppMaps system still had to be written. It is worth noting that there was no duplication of styling code at all in the Agritechnik application. If two user interface elements shared non-trivial amounts of styling, then they either were of the same type or were of the same case. The case rules and individual rules for one-off edges (specifying, for example, what icons to use for buttons) were enough to specify the styling completely.

The commissioning user also noted efficiency gains in both situations where case was used in the design process. In the first situation, the cases could be moved around the dialogue controller very easily, and took the appropriate styling with them; in context this meant that the alternatives could be rapidly mocked up within a meeting with the person commissioning the application in a way that the author would not have attempted otherwise. This is also true of the second situation: it involved more work than the previous example, but it was still feasible to do in the context of the conversation with the customer. Again, this is something that the author would have hesitated to attempt otherwise.

6.6.6.3 Expressiveness

The Agritechnik application has two main functions, both of which use case. The two functions, however, use case to differing extents. The first is a communication function not dissimilar from the 1EVENT application presented above. The user is here constructing a fairly simple thing and sending it somewhere, much as they might send an email message. There is little scope, therefore, for genitives.

In the second function, which involves navigating and interacting with data, there is greater scope. Both genitives and datives can be used here quite liberally. This, combined with the results from the 1EVENT application above, suggest that the case-based approach is far more useful for data-driven applications than for other kinds of applications.

The Agritechnik application also provided a good testbed for expressiveness across platforms. The original application was specified as running on Android and iOS; at a late stage in the development process, the commissioning person inside Agritechnik decided that they wanted the application to run on Windows tablets as well. At this time, the AppMaps system did not support Windows at all.

However, the underlying Sencha Touch toolkit *was* able to support Windows. Therefore, the question became one of whether the cases could sensibly be implemented within the Windows Modern User Interface guidelines.

Microsoft's user interface guidelines are arranged very differently from Apple's and Google's equivalents. Apple and Google both give meanings for individual components: for example, the component that splits a screen into two parts has a specific use and different kinds of lists have different uses. Microsoft's, by contrast, is laid out in terms of overall layouts for the application: they give a number of overarching patterns (such as the hierarchical pattern, the flat pattern, and the semantic zoom pattern Microsoft Corp., 2013b) and then detail how information should be laid out within those. The first stage of building a Modern Windows application, according to Microsoft, is to decide which of these patterns one wishes to use for one's application.

Case here fulfilled an unexpected function. The realisations of the cases are different in the different patterns. For example, in the hierarchical navigation pattern, categories work much like they do on iOS and Android phones, even on tablets: the list of items is on a page on its own, and the expectation is that as the screen gets bigger more details are shown. In the flat navigation pattern, by contrast, category-type meanings are shown in a bar at the top of the screen which may only be shown when the user explicitly requests it 6.22. The person within Agritechnik was not sure which of the kinds of application they wanted. Therefore, part of the design process for the Windows application involved generating stylesheets and transformation rules for both of the possible layouts. The person within Agritechnik could then use both prototypes to make the decision.

6.7 Speakers Associates: Anatomy of a success

6.7.1 Background

The third case study follows an application developed for Speakers Associates. Speakers Associates is a speaker bureau, which acts as an agent for business conference and event speakers. The application that they wanted was to display their database of speakers and let customers produce a shortlist of speakers for their events that could then be sent directly to Speakers Associates for further discussion. They also wanted users to be able to share speakers' biographies or publications on social networks. The application was to be used both by their customers and by agents in the field talking to customers.

6.7.2 Requirements

These are the requirements for the application that impacted on the user experience. As before, figures are redrawn versions of those in the original requirements document.

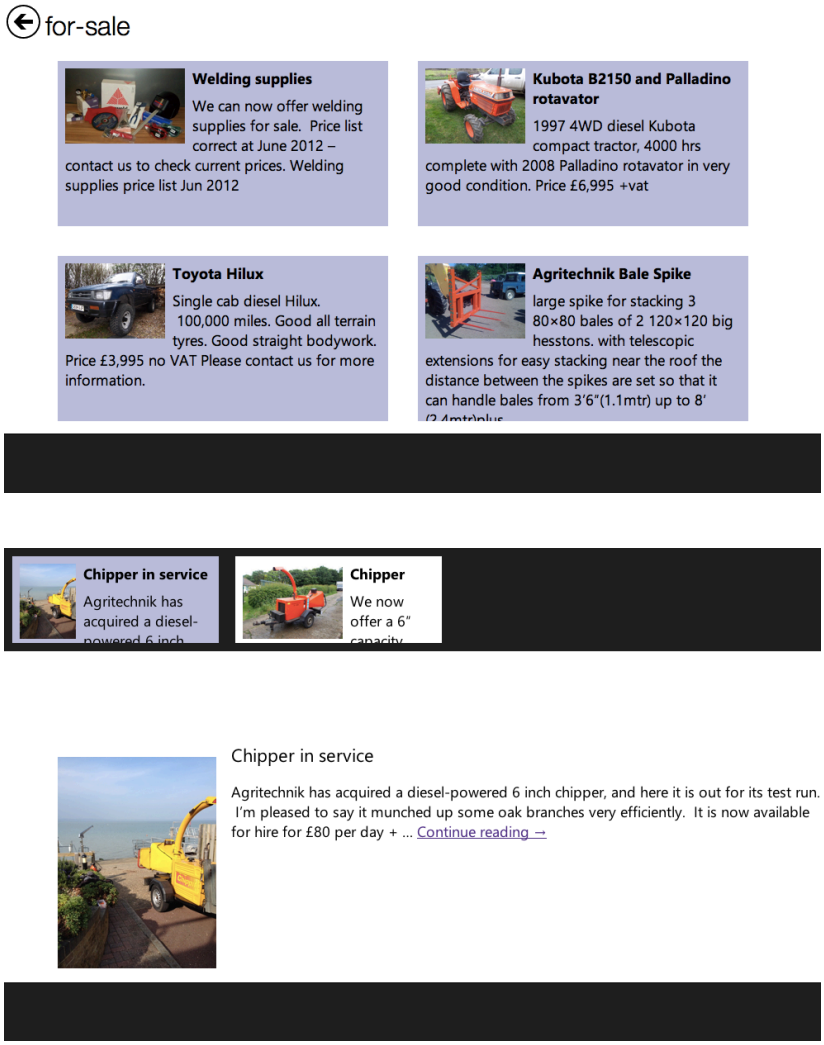


Figure 6.22: Realisations of the genitive under Windows Modern

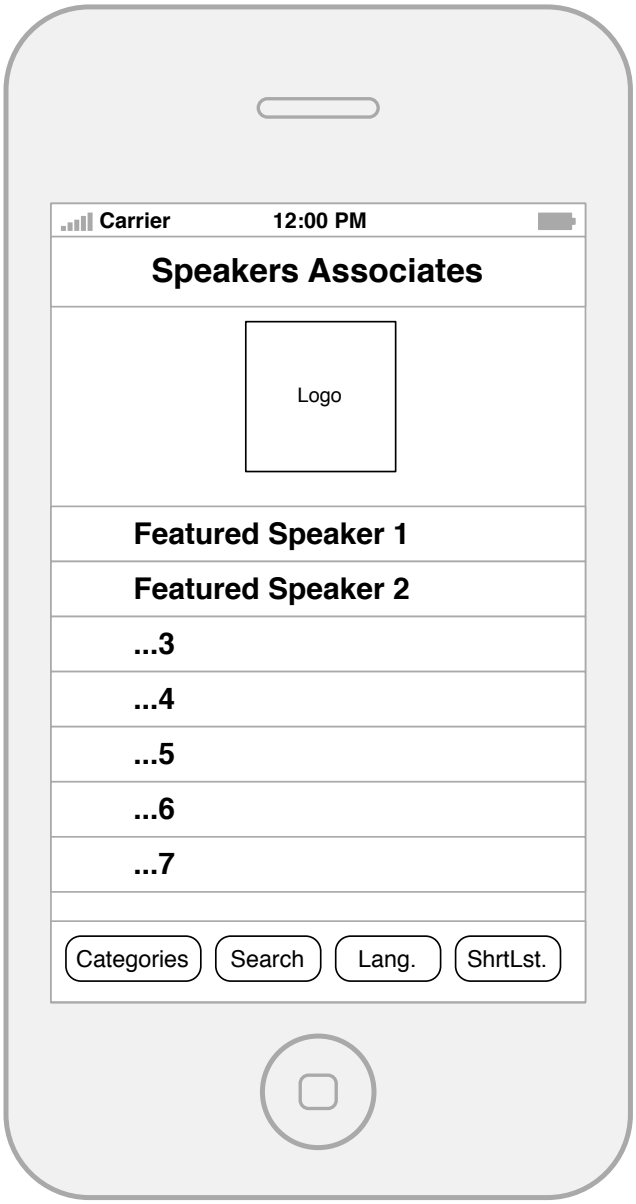


Figure 6.23: SpeakersAssociates front page

- When the user first loads the application, the first screen must contain the Speakers Associates logo, the list of featured speakers in some appropriate way for the size of screen, and the ability to view the user's current shortlist, view the speakers by category, view the speakers by language, or search by keyword (see figure 6.23).

- Whenever the user taps on the name or image of a speaker, be it in the list of featured speakers, a shortlist, a category of speakers, or a set of search results, they should be presented with a page with the speaker's photograph, name, biography, languages spoken, publications and testimonials. This screen should be the same design no matter how the user reaches it; the screen must not vary if it is invoked from a set of search results, for example (see figure 6.24).

- By the same token, lists of speakers throughout the application must be visually consistent and share the same appearance. The list of speakers in the search results, for example, must use the same physical control that is used on the front page for the featured speakers, and for the contents of a category of speakers. This control *may* differ between different form factors.

- From this screen of details, the user must be able to add the speaker to their shortlist and to share the speaker to social networks.

- The user's shortlist must show the speakers from the user's shortlist using the standard speaker list for the form factor. There should be a button to remove a speaker from the user's shortlist, and a button to contact Speakers Associates about all the speakers in the user's current shortlist.

- The option to view speakers by category should initially display a list of categories; after a category is chosen, then the speakers in that category should be shown in a standard speaker list. Categories should be identified by their textual label; there is no image associated with a category. The description of each category that is in the database should be disregarded for the purposes of the mobile application (see figure 6.25).

- The option to view speakers by language should function identically to browsing by category except that the initial list should be of languages rather than of categories.

- The option to search the speakers available should allow searching by name or keyword, as the Speakers Associates web site does (see figure 6.26). Search results should be displayed in a standard speaker list.

6.7.3 The application map

The application map for the Speakers Associates application is shown in figure 6.27.

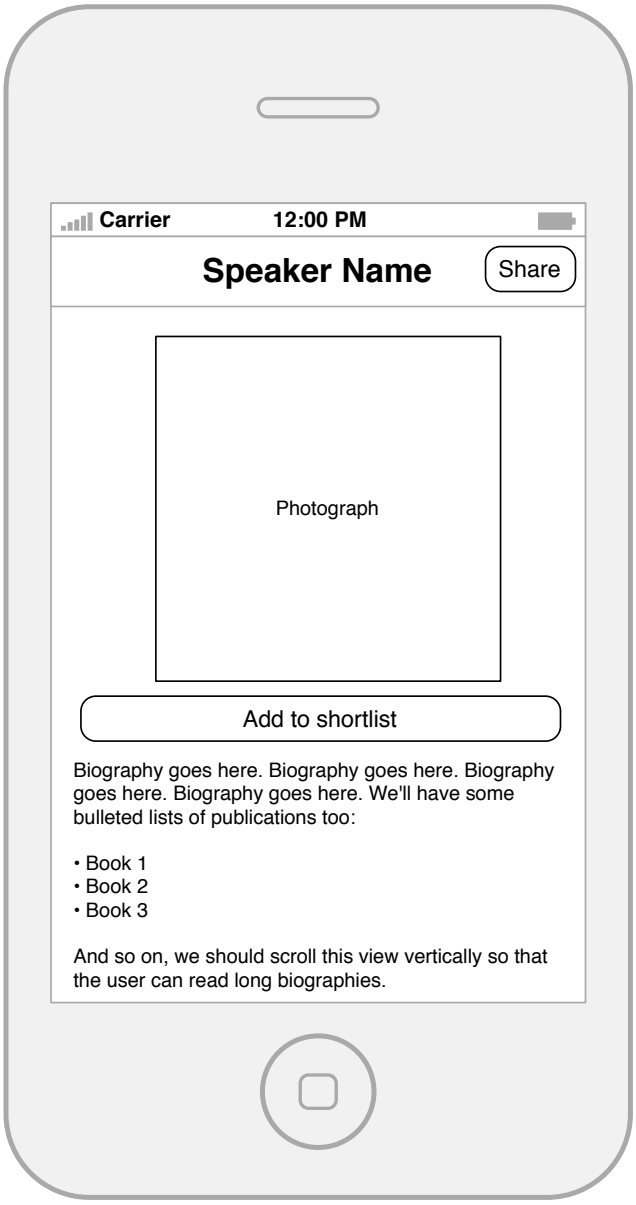


Figure 6.24: SpeakersAssociates detail page

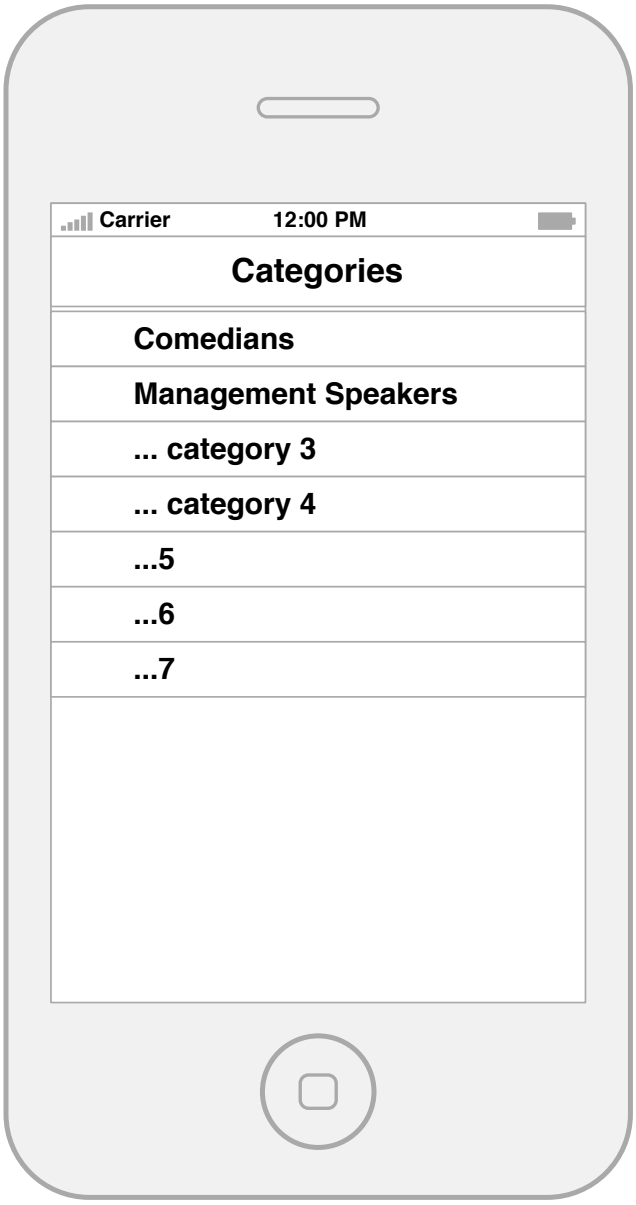


Figure 6.25: SpeakersAssociates categories page

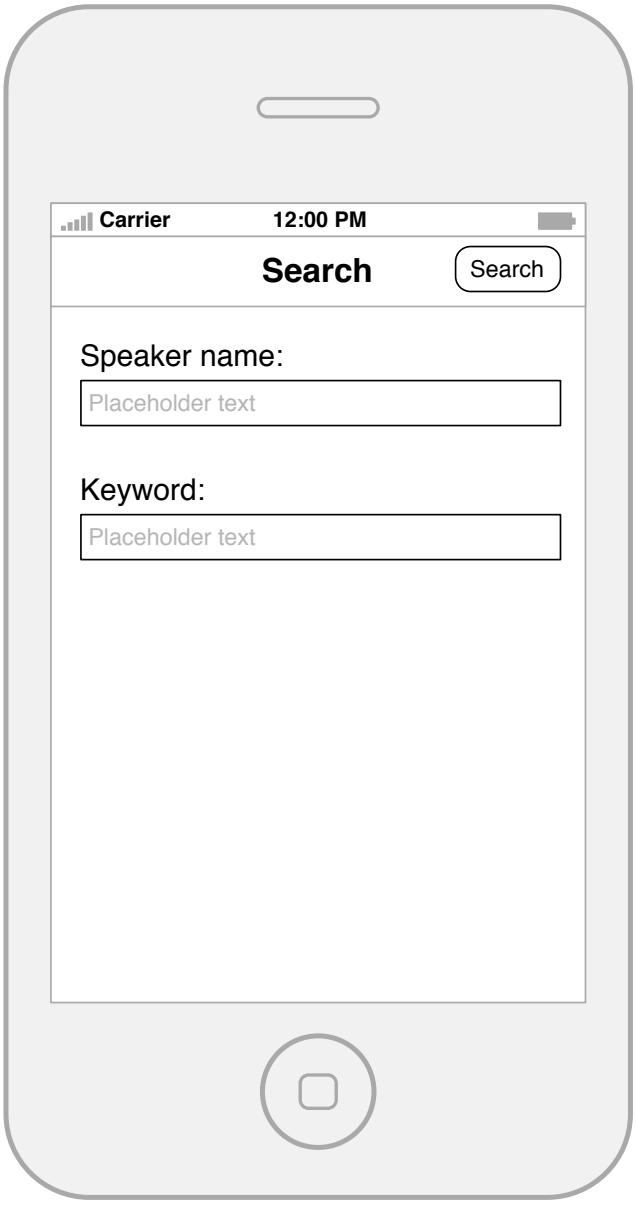


Figure 6.26: SpeakersAssociates search page

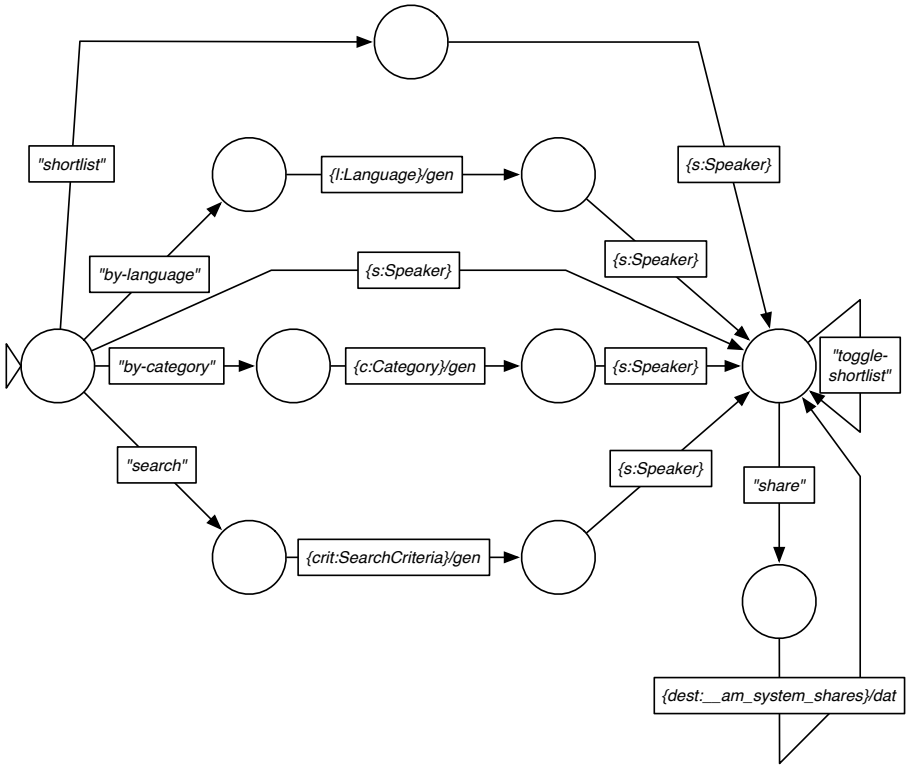


Figure 6.27: The application map for the Speakers Associates application

6.7.4 The use of cases

The genitive is used throughout the application to mark membership of a category. The application only talks about one kind of object: a speaker. The two trivial uses of the genitive are those in the category and language browsers: in these instances there is a finite set of categories that speakers can belong to, and the genitive edge selects one of these categories (figure 6.28).

The less-trivial use of the genitive can be found in the search option. The edge where the user specifies the search criteria is marked as a genitive because it, too, identifies a set of speakers by category or characteristic. If the list of keywords were finite and the user were choosing between them, then it would be a non-trivial case of using a genitive. However, since the user is expected to enter search criteria, both the author and the commissioning person inside Speakers Associates wavered as to whether this fell under the meaning of the genitive. Two alternatives were made, one with and one without the genitive marking on this. The version with the genitive marker won immediate approval (figure 6.29).

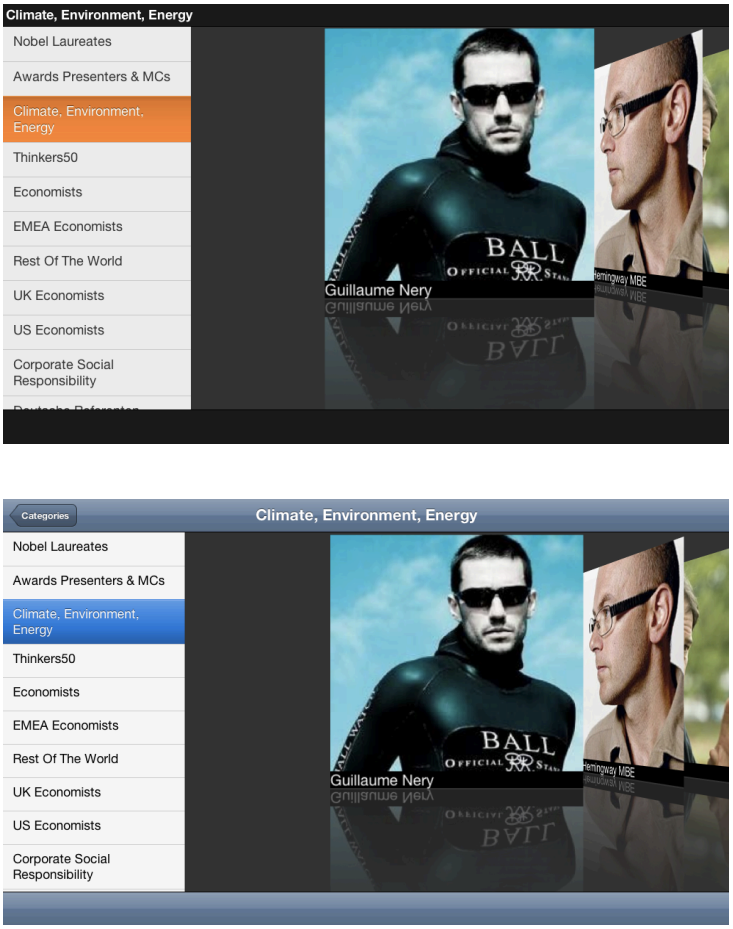


Figure 6.28: Genitives in the category view

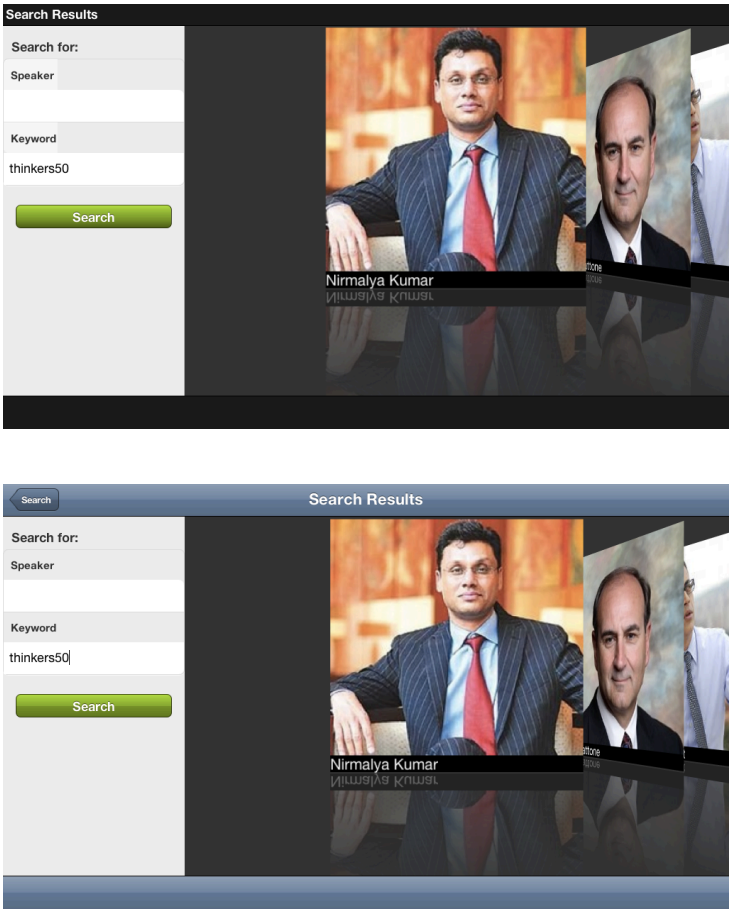


Figure 6.29: Genitives in the search view

The dative is only used in trivial cases in this application: it is used as to mark social network sharing destinations (see figure 6.30). Note that it is not used in the "Contact us" function, as there is only one destination within Speakers Associates to which all such contact attempts are directed: therefore, the user can have no choice of destination.

6.7.5 Evaluation

6.7.5.1 Effectiveness

The Speakers Associates application was developed in conjunction with one person inside the Speakers Associates company.. this person is multilingual and speaks two languages that use different case systems. This person was asked to give feedback on the quality of the application, with emphasis on its ability to adapt between platforms and form factors. As above, their remarks on the parts of the application that use case is summarised here.

The two straightforward uses of the genitive were in the screens that permitted browsing by language and browsing by category of speaker (figure 6.28 above). "These screens are fine—you want me to comment on the bit on the left on the tablet, yes? And the thing that does the same job on the phone? It's fine, but it's not the interesting bit of this screen. It's the boring bit, but it needs to be done. You've automated the boring bit. That's good." They were asked to clarify "boring", and said "It looks professional and it's how it ought to look, but it's not the bit that shows off the tablet. The nice big clear photos [in the speaker list] show off the tablet."

The less-trivial use of the genitive was in the search screen, where the search criteria were marked as a genitive (figure 6.29 above). The author's emails to the commissioning person inside Speakers Associates note that this was done "more in hope than in any real expectation of success". The somewhat laconic reply from the commissioning person consisted entirely of "It did [succeed]. Don't change it." They were asked to clarify why they thought it succeeded, and replied "It looks good. It passed the [Company employee] test, and she's hard to please. And if categories are on the left, the search criteria should be. And it lets you see what you searched for while you're looking at the results. I like it. Don't change it."

The dative was used only for social media destinations (figure 6.30 above)s. Again, the commissioning user referred to it as a "boring bit": "it's right. It's right on tablets and phones ... but it's not fun or interesting. I'm surprised this isn't already automated everywhere. It should be. You've automated the boring bits again."

On the iPhone, the "select by category" use of the genitive became two screens, the first of which showed a list of categories, each with a disclosure indicator, the

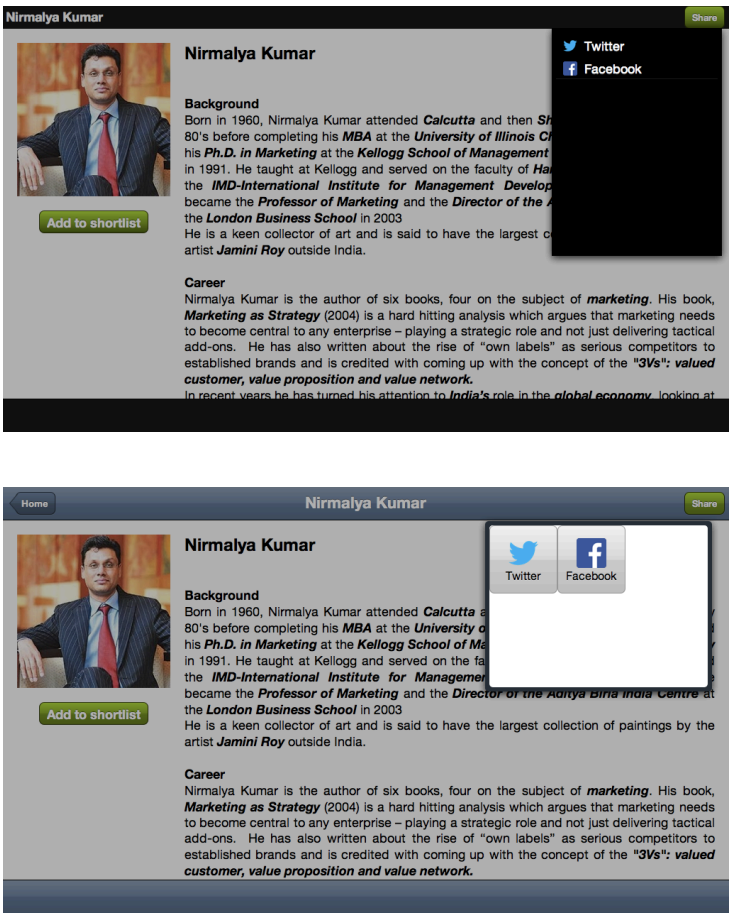


Figure 6.30: Datives in the speaker details view

latter a list of speakers in that category. In this case, the disclosure indicator is correct: the list of speakers is a table view (HIG p. 170). The "select by language" use of the genitive worked identically.

On the iPad, these genitives became split views, as in the Agritechnik application. Again, these were non-controversial according to the platform guidelines. The left panel contained the list of categories, and the right panel contained a "cover-flow" view that showed the speakers in that category. This is an appropriate use of the split view control.

On Android phones, the realisation of the "select by category" and "select by language" genitives was very similar to the iOS realisation. As in the Agritechnik application, no disclosure indicator was added, and so this realisation met the requirements of the Android human interface guidelines.

On Android tablets, the realisation of the "select by category" and "select by language" genitives became multi-pane interfaces similar to that for the Agritechnik application. Each has a list in its left panel and a "cover-flow" view in its right panel.

The idiom that both the author and the commissioning user had more concern over was the search. On the iPhone, this became two pages, the first containing the search form and the second the search results. On the iPad, this became a split view, with the search form in the left panel and the search results, after the search was complete, on the right. This, although seemingly unorthodox, is explicitly permitted by the iOS guidelines, which state that the split view supports "table, map, text, web or custom views". On Android, this is likewise compliant with the human interface guidelines, as they make no statement whatsoever about what views should be in use in the different panes of the interface.

6.7.5.2 Efficiency

The Speakers Associates application provides useful evidence for the efficiency gains of using case, because an initial prototype of this application was built using a pure Sencha Touch and PhoneGap software stack, so actual comparison of the development processes involved in building the two versions is possible.

The user interface for the Sencha Touch version of the application consisted, as far as the software was concerned, of a viewport in which all the screens of the application were embedded. The set of screens was the same as in the AppMaps version of the application. The Sencha Touch 'card' layout manager was used to make sure that only one screen was visible at once.

The requirements of the application mandate a certain level of logical adaptation: specifically, the list of speakers should use an "appropriate" kind of display to the size of screen in use. Since this adaptation doesn't depend on what context the

list of speakers is being used in, it cannot have anything to do with case. Therefore, any software components that are only responsible for this logical adaptation cannot be used to argue for the efficiency gains of case.

This leaves the software components that are necessary to perform the physical and dialogue adaptations. In the AppMaps version these are entirely covered by the dialogue controller and the stylesheet. In the Sencha Touch version, two versions were made of each screen that needed to adapt, except for the sharing palette. One of these versions covered tablet platforms, one covered phone platforms. The views that needed to be adapted in this way were the ones that let the user select speakers by category and by language and the one that let the user search the database.

This suggests that, as indicated by the Agritechnik example, the AppMaps version requires a fairly constant number of software components to effect plasticity, whereas the number required in the manual approach scales linearly. However, a lot of this is simply due to having a centralised dialogue controller. If a half-way house between the two applications had been developed using an AppMaps-like dialogue controller mechanism but with no case, then there would have needed to be one dialogue map for each platform and form factor, or at least one for each set of conventions in use. This is still scaling linearly, but is doing so in the number of platforms and form factors, which is unlikely to grow in quite the same way as the complexity of the application.

The author's subjective impressions of the efficiency of the tool on this project were largely centred around the economy with which the two categories of dative and genitive captured the structure of the *data* that backs the application. The commissioning person inside Speakers Associates was both multi-lingual and wanted to be highly involved in the details of the development process. When they understood that the concept of case was the thing that was under evaluation, they pushed to see where it would be useful in the development process.

This lead to the use of case at the data modelling stage of the project. Even before the user's paths through the application had been decided, the links between the entities involved were marked with cases (figure 6.31). For example, the link between "Languages" and "Speakers" in the diagram shows it to be a genitive relationship. From this, the application development process for the AppMaps version of the application became almost an extremely lightweight pencil-and-paper model-driven engineering process.

This made the core of the application map nearly a foregone conclusion: the parts of the application map that did not emerge directly from the data model were the communication functions (sharing to social media and contacting the company) and the featured speakers list. This made actually designing the application map, complete with cases, very quick indeed.

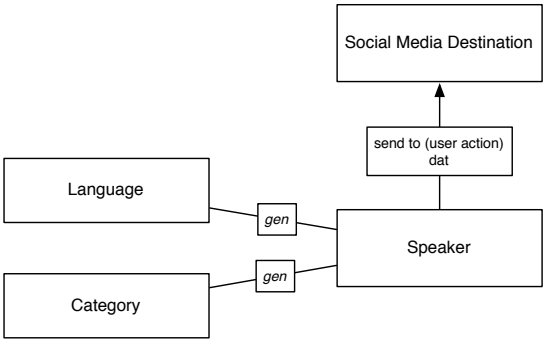


Figure 6.31: Core data structure for Speakers Associates application

6.7.5.3 Expressiveness

The Speakers Associates application is primarily concerned with navigating and exploring a data set. Case played a useful role in this part of the application. There is a tiny communications function in the allowing of the user to send a message back to Speakers Associates: case was not at all useful in this part of the application. This reinforces the suggestion from the first two applications that case is more of use in data-driven applications. There was nothing in the application domain that made the application of case difficult.

The only platforms initially required were iOS and Android on tablets and smartphones, and the case system captured the dialogue patterns that were required for the application.

In fact, the Speakers Associates application made the author aware that the case system of AppMaps is actually slightly more expressive than had been expected, largely due to the use of the genitive for search criteria.

6.7.6 Overall feedback from clients

The three companies for whom the applications were developed were asked to provide a brief comment on the development process and the resulting application and how well it had addressed their needs to summarise their previous comments. The commissioning person at 1EVENT said: "I'm aware our app didn't use most of the special features of your tool. The application was good, but I'm not sure how much good we got out of it that we wouldn't have got out of a more conventional cross-platform framework." The commissioning person at Agritechnik, whose app was more successful, commented that "[t]he main benefit to us was the flexibility, I think. We are not a digital technology company and we were, and still are, experimenting with where an app could be of use." Speakers Associates, whose app was nearly an ideal fit for the tool, said "The most useful thing was the

speed. The app was good, and you built it very fast. As I said before, the tool took care of the boring bits."

6.8 Summary

The table below summarises the conclusions from the case studies.

Effectiveness		
EVENT2	Agritechnik	Speakers Associates
Commissioning person found application to be at home on all desired platforms. Marketing person said it looked strange but identified this as a problem with HTML5 applications as a whole. Dative, the only case used, rendered correctly according to human interface guidelines on phone and tablet form factors both under Android and iOS.	Commissioning person found application to be at home on all desired platforms, "as far as I can see. I'm not the expert on mobile apps here." He also considered the application running on Windows Modern UI to be good quality. Both dative and genitive rendered correctly according to human interface guidelines on phone and tablet form factors both under Android and iOS.	Commissioning person found application to be at home on all desired platforms. He considered the tool to be "automating the boring bits" and leaving energy for "more interesting" design decisions. Both dative and genitive rendered correctly according to human interface guidelines on phone and tablet form factors both under Android and iOS.

Efficiency		
EVENT2	Agritechnik	Speakers Associates
When only the application considered, minimal efficiency gain over a pure Sencha Touch approach.	Number of software components effecting plasticity was low and remained constant as new platforms were added. By contrast, in a vanilla Sencha Touch approach the number would have scaled linearly with number of views that needed to adapt, and in Sencha Touch with a dialogue controller, the number would have scaled linearly with the number of platforms. Was unexpectedly useful as a way of exploring alternative versions of the interface with commissioning person, both in adding a new platform and rearranging the existing dialogue machine. Developer's subjective experience suggests that it was quicker to use to develop than either pure Sencha Touch or other systems with dialogue controllers (Apple XCode, NetBeans J2ME toolkit).	Number of software components effecting plasticity was low and remained constant as new views were added. No new platforms were added in the course of the development process. A previous pure Sencha version of the application confirmed that the number of software components effecting plasticity in that framework scaled linearly with the number of views that needed to adapt. Case unexpectedly made available a lightweight paper-and-ink "model-driven engineering" approach by being used to describe data as well.

Expressiveness		
EVENT2	Agritechnik	Speakers Associates
Industry is primarily mobile communications.	Industry is agricultural equipment servicing, sales and hire.	Industry is event management and speaker agencies.
Purpose of application is communication.	Purpose of application is partly communication and partly data browsing.	Purpose of application is data browsing.
Encompassed platforms that commissioning company wanted.	Encompassed platforms that commissioning company wanted, including an unexpected requirement for Windows Modern UI introduced late in the development process, though this did require several grammars for different styles of Windows UI.	Encompassed platforms that commissioning company wanted.
Did not require the dialogue plasticity overriding at any point.	Did not require the dialogue plasticity overriding at any point.	Did not require the dialogue plasticity overriding at any point.
Case was more expressive in this instance than the application required: the genitive was not used at all.		Unexpected expressive by correctly plasticising search criteria in search function.
		Commissioning user, having an explicit awareness of case, found it a useful vocabulary to use to describe the application.

6.9 Conclusions

6.9.1 Effectiveness

The first criterion for effectiveness given in chapter 4 is:'Is the commissioning user happy with the application on all platforms and form factors?' The commissioning users of all three applications felt that the applications were at home on the platforms they wanted to use. The commissioning user of the Speakers Associates application noted that what was being automated was the "boring" and formulaic parts.

The second criterion for effectiveness is: 'Does case contribute to the application's conforming to the human interface guidelines for the desired platforms and form factors?' From the evidence given by these three case studies, the answer is yes, but with caveats. Certainly the screens that used case to determine their appearance conformed to the human interface guidelines.

However, case by no means covered all the ground that the user interface guidelines cover. The Apple iOS human interface guidelines, for example, filled a 220 page book as of the end of 2013. The rules that guided the implementation of case, and the rules that were used to evaluate the resulting applications, were contained on fewer than ten pages of this book.

An equally telling fact can be found in the Android human interface guidelines. Android has suffered somewhat from badly ported applications from other platforms, as the "Pure Android" section in the human interface guidelines shows. This section gives six specific common problems that cross-platform applications have (these are directly quoted from the "Pure Android" section):

- Don't mimic UI elements from other platforms
- Don't carry over platform-specific icons
- Don't use bottom tab bars
- Don't hardcode links to other apps
- Don't use labelled back buttons on action bars
- Don't use right-pointing carets on line items

Of these, the only one that has any bearing on the dialogue layer at all is the using of labelled back buttons on action bars. The proscription about hardcoding links to other applications is a technical one. The other four are all about physical adaptation. Case can have no input into any of these conundra.

6.9.2 Efficiency

The first criterion for efficiency given in chapter 4 is: 'How many software components are involved in effecting plasticity? Is case responsible for any reduction compared to the other comparable industrial tools?' In the case studies, efficiency gains were demonstrated. It is worth noting that none of the applications described above adapted at any level above the dialogue layer on the Arch; there was no functional core adaptation for different platforms.

In a pure MVC approach with no dialogue controller, the number of software components that effected plasticity scaled linearly with the number of views that were affected by the need for plasticity. In the EVENT2 and Agritechnik case studies, this was worked out hypothetically. In the Speakers Associates case study, this was confirmed from actual software. In an MVCD approach, or some other approach with a centralised dialogue controller, then this is reduced to being linear in the number of platforms and form factors. In a system with a dialogue controller that can successfully adapt, for an application such as these where no functional core adaptation takes place, the number of software components effecting plasticity is constant or near-constant.

Case was the sole means by which the dialogue component could adapt, and provided one of the major mechanisms for logical and physical adaptation also. Since this was so, and since in the case studies the number of software components effecting plasticity was near-constant and did not scale linearly either with the number of views involved or the number of platforms, it is reasonable to ascribe this efficiency gain to the use of case.

The second criterion for efficiency is: 'Does the use of case provide a tangible benefit in terms of time at any specific point in the development process?'. Two of the case studies suggest the usefulness of case for exploring alternative interfaces during the design process. Both the Windows UI for Agritechnik and the search criteria for Speakers Associates are useful examples for this, because they both hinged on the idea of building alternatives that 'mean the same thing' (as the commissioning person at Speakers Associates put it). The meaning in these two instances was described by the cases and the labels on the kind-of-object edges: and the different realisations of these meanings were what was being decided between. The efficiency gain in the design of the advertisement browser for Agritechnik was simply due to having a rapidly reconfigurable dialogue controller.

6.9.3 Expressiveness

The first criterion for expressiveness given in chapter 4 is: 'Does it cope well with the platforms and form factors that the commissioning user needs?' In these three cases, case did cope well with these platforms and form factors. Primarily, the

requirements were for Android and iOS tablets and phones, although Agritechnik also requested Windows Modern. That case is expressive enough to deal with the idioms of these platforms is suggested by the fact that in none of these applications was it overridden: none of the application maps devolved to being a cluster of maps maintained by hand, one for each platform and form factor pair.

The second criterion for expressiveness is: 'What kinds of applications can case be useful for?' The case studies contained one application that was purely designed to let people communicate, one that was purely designed to let people browse data, and one that had components for both of these functions. As outlined above, the application that used case most successfully and pervasively was the purely data-driven one. The application that integrated both functions used case most successfully in its data-driven aspect, and the pure communication application found little use for it. It would not be surprising if this were true, because case is fundamentally about the connections of objects with other objects, and there is more scope for this kind of connection in a data-driven application than one that does not use complex data. This suggests that case can be far more useful for data-driven applications than for communication applications, and gives something to watch for in the results of the studies with other developers.

The third criterion for expressiveness is: 'Is case more useful for some industries than others?' The case studies addressed problems from three very distinct industries. No evidence emerges from them that case is better suited to any given industry segment than any other. The comparative failure of the EVENT2 application is better explained by its being a communications application than by it being written for the mobile communications industry.

The three case studies described here provide suggestive information about the usefulness of case, but they are limited because they are all projects that the author undertook. The next chapter addresses this shortcoming by examining other developers' reactions and use of the AppMaps tool and, through the tool, the case system.

Chapter 7

Studies with other developers

7.1 Introduction

In chapter 6, a number of applications were described that had been developed by the author of this thesis for different clients and different industrial situations. The usefulness of case was analysed in that context through an examination of the development process and the clients' reactions to the finished applications.

As noted in chapter 4, an evaluation involving independent developers is needed for drawing any conclusions about the general usability of case as embodied in AppMaps. To that end, two studies were undertaken involving, in total, fourteen developers. The present chapter presents the details and results of these studies.

The first study was a workshop study, in which a number of developers performed a role-play exercise in which the author played the part of a client. The participants then developed an application map that met the client's requirements. They did not have to write any source code during this exercise. Instead, the emphasis was on the use of the case system and, to a lesser extent, of the application map. In this exercise, developers drew on their professional experience to talk about the usefulness of the tool in a broad sense. They also provided seeds for iterative design for a second version of the tool for the second study.

The second study was a self-directed development study. In this study, developers were given the tool to use for a task of their own choosing. They were provided with documentation on the tool and the case system and provided with technical support by the author, but no constraints were made on their development process or on the kind of application they chose to make. When they either completed their application or could devote no more time to its creation, they provided both the application and some discussion around their development process. This provided more detailed information about the model's usefulness in concrete circumstances, and paralleled (although with smaller projects) the case study approach in chapter 6.

These studies were mostly aimed at providing information for the third and fourth research questions, about whether case can be used to build useful applications and whether case can be used by other developers. The two evaluation studies follow the same three Es as were used in the previous chapter, but looked at from another developer's point of view. Effectiveness therefore is about whether other developers managed to produce user interfaces that they considered high-quality; efficiency about whether other developers would be more productive using the case system than not; and expressiveness about whether other developers could build the kinds of application that they wanted to build using the tool and the case system.

7.2 Goal and approach

The goal of the work presented in this chapter was to provide a wider perspective on the usefulness of case to the commercial development of plastic user interfaces. This is necessary, as noted in chapter 4, to complement the case studies presented in chapter 6. The case studies only concentrate on a single developer—the author—who may be atypical, and concentrate on the details of individual applications. The work presented below adds two extra layers to this: the self-directed development study widens the perspective to collect information about other developers building specific applications, and the workshop study widens the perspective still further with other developers considering how the use of case would fit into their general professional practice.

Again, it is important to distinguish between the *research approach* that is used to gather information and the *development approach* that is used, in this case to develop the tool. The development approach was iterative: the workshop was used to find issues—be they usability issues or software quality issues—with the tool before it was given to developers for the longer development study.

The research approach used was a co-operative evaluation approach, which was explicitly designed to fit well with an iterative design development approach. In the two archetypal co-operative evaluation exercises presented by Wright and Monk (1991), users were given a task and a manual for the system, encouraged to think aloud while using the system, and encouraged to consider themselves as evaluators of the system. In the case of the two evaluation exercises presented in this chapter, the task was to build an application using the system.

7.3 Workshop study

7.3.1 Introduction

As outlined in chapter 4, the first of the two studies that involve other developers was a workshop study. This workshop had two primary goals. The first was to gather developers' feelings about the tool as it applied to their professional practice in general. This goal was approached through a number of questions, which were given in chapter 4. The second goal was to find usability issues and software defects in the tool itself, so that a second iteration of the tool could be given to the developers in the longer study.

7.3.2 Method

7.3.2.1 Participants

Seven participants (who will be referred to henceforth by the letters A to G inclusive) took part in the workshop. The participants were recruited through the author's professional network.

All participants except participant C were developers to a professional standard, and all but participant C were developing for mobile platforms outside the workshop. These six were all either targetting Android, iOS or both. Three had used versions of the Apple development tool that include Storyboards (see section 2.8). One had used the Netbeans J2ME mobile toolkit (also described in section 2.8). Four were using existing cross-platform tools . Five identified as knowing JavaScript well. Participant C was a hobbyist developer.

7.3.2.2 Apparatus

A laptop was provided for each pair of participants. Each laptop was running an identical virtual machine so that the experience of each participant was as near identical as possible. This virtual machine contained Windows XP, the AppMaps tool, a text editor, the sample projects to be used for the workshop, and a mobile device simulator. As there was an odd number of participants, participant A had a laptop to themselves. Each pair also had a 'cheat sheet' which contained a brief description of the stylesheet syntax and a definition of each case. Each case definition contained the same information as the definitions in chapter 5 rendered as bullet points.

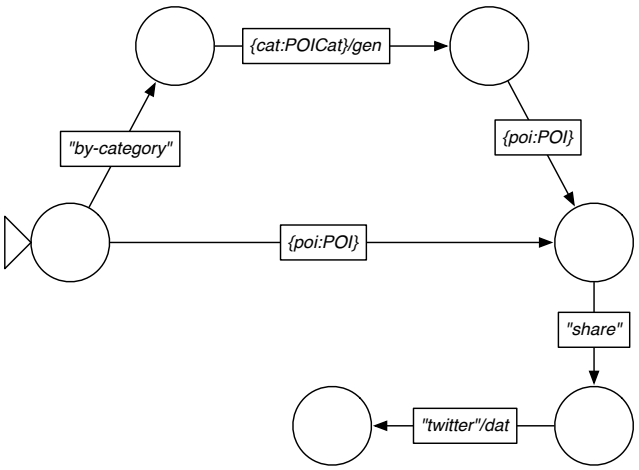


Figure 7.1: Satellite navigation application map

7.3.2.3 Procedure

All data gathering was undertaken during a single workshop session during which participants A to G were co-located in a workshop room. Both video and audio were recorded for the entire workshop session.

The workshop was split into four phases. In the first phase, participants were shown how to create a simple satellite navigation application. The application map of the application is given in figure 7.1. Initially, participants were walked through the tools used to create an application map, and how they could draw out nodes and edges. After this they were introduced to the difference between one-off and kind-of-object edges, and were walked through creating the application map without any plasticisation or use of case. Once participants had created that map, they were given an overview of the styling language and shown how to change the visual appearance of buttons in detail. Finally, they were shown how to attach cases to edges, how this affected the resulting application, and talked through the definitions of the cases on the cheat sheets.

The second phase took the form of a question and answer session, where participants were encouraged to raise any issues they'd had with the use of the tool in the first half, and any points on which they required clarification. This phase was freeform, and based around issues that participants had experienced.

In the third phase, the participants were given a series of requirements to build a small e-commerce application for a wine shop. The author had already built a model answer application, but this was not shown to participants. The requirements were revealed in small sets, so that the participants remained in step with one another. During the design process, the author walked around the room to

check on the progress of each group and to answer any questions that participants might have.

The first set of requirements were (quoted verbatim from the materials given to participants):

- A list of featured wines on the front page of the app.
- Buttons to let the user choose by country or by colour.
- When a country or a colour is selected, let the user select wines from that country or colour."

The second set were (likewise, quoted verbatim):

- A 'contact us' button on every page of the application
- They say the form will be designed later by their developers...
- If you get time, try making the 'contact us' buttons red.

At the end of these two sets of requirements, participants would have a simple phone application that was not plasticised.

Further requirements involved using case to plasticise the application. The fourth set were:

- On a tablet, the colour and country selections should be in a two-column view.
- Hint: use cases.

Finally, the fifth set were:

- A "share" button on each wine to send the wine details to someone else.
- This button should pop up a box to let the user choose what service to use.
- For now, they just want twitter.

After participants had finished addressing these requirements, the author went around to each and compared the resulting application map to the model answer. Where there was a deviation, participants were asked what their rationale was for designing that part of the application map.

In the fourth phase a series of discussion prompts were presented. Four prompts were presented for each of effectiveness, efficiency and expressiveness.

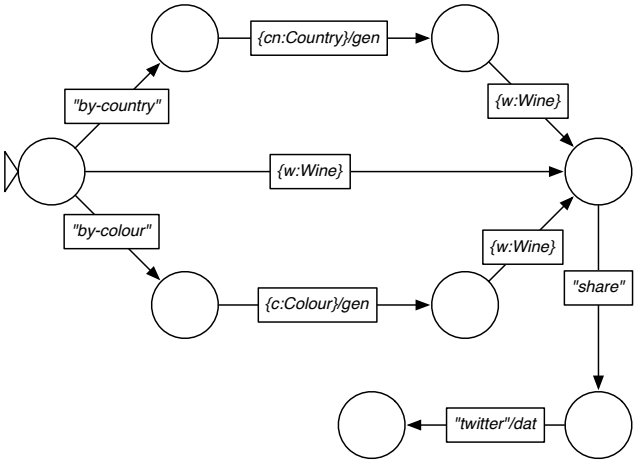


Figure 7.2: Wine shop application: model application map

The first discussion prompt for each was the word itself, with a brief description of how it was being used in this context (as in chapter 4). After this, each of the questions raised for that word in chapter 4 was presented for discussion.

At the end of the discussion period, participants were asked for their criticisms and opinions with regards what ought to be changed in the tool for the next version. This part of the discussion was divided into two parts, one for the fundamental model, and one for the user interface of the tool itself.

Finally, the software that participants had created was collected and archived. No JavaScript was written as part of the workshop: the purpose of it was to examine the usefulness and comprehensibility of the case system, rather than evaluate the participants' abilities to write JavaScript.

All remarks from the workshop that bore on case were transcribed, along with the entirety of the discussion section.

Ethics approval was granted for this study.

7.3.3 Results

7.3.3.1 Application map variations

The model answer application map for the wine shop is given in figure 7.2, although the "contact us" edges are omitted in this figure for the sake of clarity. This model answer was not intended to be a *correct* answer as such, but more the author's expert opinion on how the application might be constructed.

Two of the pairs of participants produced different application maps from the model. The application map most divergent from this one was produced by par-

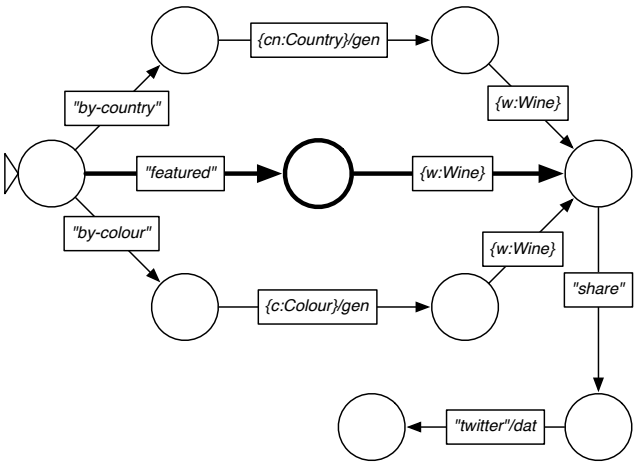


Figure 7.3: Wine shop application: participants B and C's map

ticipants B and C and is shown in figure 7.3, in which departures from the model answer are shown in bold. There are two differences from the model.

- The model application had the list of featured wines on the front page of the application. Participants B and C put a button on the front page that the user had to press to get the list of featured wines. Their given reasons for this were aesthetic: they would have wished to have a "picture or branding or something" on the front page of the application.
- The model application had a "Contact us" edge from every state. Participants B and C had removed all these edges because they were making the map "too crowded".

Participants D and E had diverged from the model in a smaller way. Their application map, given in figure 7.4, did not have a separate share button as requested by the "client", but instead embedded the twitter button directly into the view that would have shown the details of the wine. The reason they gave for this was that it looked "silly" having only the one share destination in the activity palette, and they would have urged the client to have the twitter button embedded directly on the page. The remaining three participants produced an application map that was structurally the same as the model.

7.3.3.2 The use of case during the application-building process

In the requirements given in the workshop, the building of the initial phone application and its subsequent plasticisation were two different phases, with a fairly

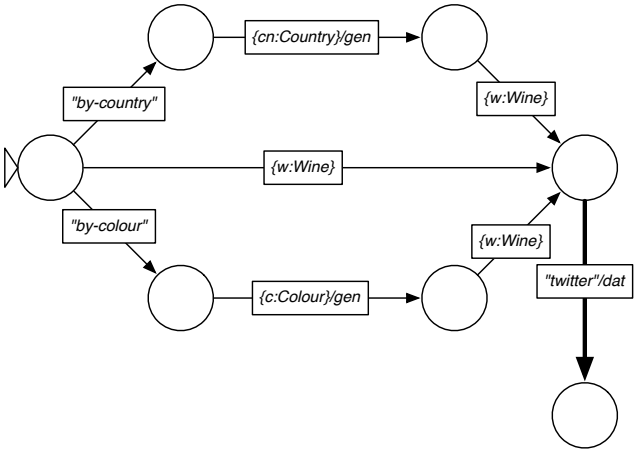


Figure 7.4: Wine shop application: participants D and E's map

heavy hint being given to participants that they should use case in this process. In practice, this does not mirror what the participants actually did.

In fact, by the time they reached the stage where the requirements were asking them to use case to plasticise the interface, every group had already done this plasticisation. As they had added each edge to the application map they had marked it with a case. Participants B and C were unusually vocal about their decision processes:

> Participant C: Oh, do we need to make this a category?
> Participant B: That'd be /gen

7.3.3.3 Points raised during discussion

The entire discussion is laid out here. No points from the workshop have been omitted.

Effectiveness The first question prompt was just the word "Effectiveness", and an explanation of the word in the sense in which it is used in this thesis. This prompted immediate discussion about what a "high-quality" plastic user interface was in the context of mobile applications. Participant G (who was very vocal throughout the workshop) started this line of conversation off:

> *The gen and dat stuff works well but (pause) it doesn't address all areas of plasticity. I mean (pause) I was building an app [not using AppMaps] where it looked fine on a Galaxy S3 but it didn't on an S2 because it didn't do the text scaling properly and it's those bits of plasticity, or even not really plasticity, but it's those bits that seem to suck up most of my time, or something. I dunno.*

Participants then identified a number of other areas other than those that AppMaps addresses that impact severely on how they perceive the quality of a cross-platform mobile application: areas identified were text rendering, the precise nature of gestures, the characteristic curves of "inertial" scrolling, graphics anti-aliasing, typography, quality of icon design, and screen layout. Participants D and E noted, to subsequent general agreement from the group, that shortcomings in these areas were not specific to AppMaps but seemed to be general shortcomings of cross-platform tools.

Discussion then moved on to whether the developers were happy with the applications that they had produced. Again, participant G led off the discussion:

> *90% of the apps on the app store look exactly like that (gesturing to screen with phone and tablet apps both displayed)*

Participants A, B and C were in agreement that given the limitations in user interface quality previously discussed, they were happy with their results, and that the case system had helped them get to results that they liked. Participants D and E noted that they would have had to have added new views to the project to get the interface that they had wanted (see section 7.3.3.1 above), but that they had not been told how to do this in the workshop itself.

Following this, discussion moved on to the topic of where the tool and the case system fitted in the problem of plasticity. Participant B pointed out that "an awful lot [had] already been said" about the scope of the problem and the tool's part in it in the earlier discussions about what made a high-quality user interface, mostly by enumerating the things that the tool didn't do. Participant F explicitly agreed with this, and pointed out that participant G had already "put it very well" previously in saying that the case system worked well but was an extremely partial solution to the problem.

The final area of discussion covered the area of whether the participants felt, based on their experiences in the workshop, that the tool had a place in their professional practice as developers. Participant B began the discussion:

> *Yes, if you fix it.*

The author's enquiring about what he meant by "fix" prompted participants to provide a number of issues with the tool that would prevent them using it effectively. The issues they raised were:

- The text fields that appear when the user edits the label on an edge were not sufficiently differentiated from simple display text, making it difficult to tell whether the user had successfully begun editing the label.
- No syntax checking was done on labels at editing-time, making it easy to create edges with illegal labels that would prevent the application from working in the simulator. Specifically highlighted was an issue where it was possible to press enter at the end of a label which, instead of cancelling the editing of the label, would put an invisible carriage return at the end of the label, which the run-time system of the application would then raise an error about.
- The "Quit" menu item did not work.
- Some mechanism for grouping states would be useful: either a statechart-like approach or by being able to show and hide subsets of states (a suggestion made by participant E).
- There was a software bug in which the map processing for dative edges did not produce the correct results.

Participant F was explicit that none of these things touched on the case system itself and was agreed with. Aside from the software bug mentioned above nobody highlighted any problems with the case system. The group finally agreed that participant B's initial statement was a fair statement about the tool as a whole with the exception of participant C who said she had no professional development practice so the tool could not fit in with it.

Efficiency Initially, again, discussion about efficiency started with participants being prompted just with the word and an explanation of its specific meaning in the context of this thesis. Participants' primary—and unsolicited—reaction was a feeling that attempting to *measure* developer productivity was a "dubious" (participant G) undertaking.

The discussion then moved on to gathering, as planned, subjective opinions about the potential efficiency gains of the tool on developers. The first discussion prompt was about whether development would be more efficient using the AppMaps case system, and under what circumstances. Participant B initiated the conversation by saying that it would provide a major efficiency benefit when they were building an application where the actual logic was complicated and they wanted to have some of the load of building the user interface lifted off their mind:

> *To be honest, when you're concerned with, [like things that] I do, something new, [where] there's very little help online, you tend to get tied up in how to*

get the guts of that to work, and you end up with an app that is essentially an implementation of something novel rather than user-focussed because I just don't have the time, but, yeah, like that, you can see what the finished product is going to look like. User-centred design, I suppose

Other participants agreed that the tool made it feasible, at least in the examples they had been given, to separate off thinking about dialogue plasticisation from thinking about the individual components in the interface, although participant G cautioned that sometimes this is such a small concern that they would just ignore it:

Whether the gen and dat are useful depends on how much you care whether your app is device-specific, really..

Participant E noted that for them, the case system would cause the maximum efficiency gain in the situation where the developer is thinking first and foremost in terms of the data that they're manipulating, and would like the dialogue component of the user interface to follow that data structure in very conventional ways. Participants A and D explicitly agreed with him.

The conversation then moved on to whether the case system provided enough of a benefit to be worth learning, given that it was an unfamiliar structure from other development systems. Conversation initially circled around the terminology, and the obscurity of the words "genitive" and "dative". None of the participants said that they had found the terminology particularly difficult: participant E pointed out that they worked in the application area "which gave the world the words byte and nybble". Participant D agreed, saying

I get what gen does, I just don't know where the name comes from though [...] but there's a lot worse technical gibberish out there.

Participant A pointed out that he had found the case system entirely intuitive. Two participants reacted to this by noting that they had found the initial discussion of the case system somewhat baffling but had understood it rapidly once they'd started using it.

C: Once we got into doing it ourselves it's straightforward and ... intuitive I think. Now that we are where we are now, building this thing, it's very intuitive. For me, it [difficulty in using the tool] is more about the labelling. I understand it, I get it, it's just the terminology I don't understand. Heh, [B's real name] is really into it now.
B: I am, I am.

Participant C had earlier said:

> *I'm at a drag and drop level... and you have these names, I'm really, what's that even mean to me? I'm at the level of, just put things in lay terms. [...] yeah, no, I get it. I've got instructions and even if I didn't I could get the hang of it, yeah.*

Participant F said that while he had understood the meaning of the cases, he didn't feel that he had really understood what was going on in software terms until he'd worked out what the cases were doing to the application map.

Expressiveness The conversation about expressiveness began with just the word "expressiveness' used as a prompt, and a brief explanation of the word as it is used in this thesis. Conversation among the participants began with participant D suggesting

> *Couldn't this be most of what is needed for most of the apps out there? There's very little actual logic in them.*

This immediately lead onto discussion of what kinds of applications the case system of the tool would be useful for building. There was agreement that a lot of applications fit the template of being centred around a remote data set and being able to browse and manipulate it. E-mail and instant messaging clients were raised by participants D and E as being obvious candidates aside from the e-commerce type applications that had been explored in the workshop. Other participants provided other application areas that were not simple browsers for data sets, but carried on a similar "basic model" (participant D). These were media players and streamers (examples given were Netflix and Spotify); personal organisation applications such as to-do list applications, diaries and book cataloguing applications; and the online browsing of reference books.

This line of conversation was disrupted by a mild dispute that began between participants F and G as to whether it would be useful for all applications or merely some of them:

> *F: It's obviously useful for a very interesting subset of apps, I mean, without even thinking about it too hard or the subset of apps ... these data selection and filtering apps, I mean, but there's no reason why it couldn't expand to other apps.*
> *G: I don't see that as a limitation, myself.*

Participant G, and with him participants A and E, thought that the case system would be worth attempting to apply to any application that they wanted to build. Participants F and D thought that it would be more use restricted to applications that had an explicit data model. Participants B and C explicitly did not wish to participate in this discussion.

At this point, the conversation moved on to whether any specific areas of application would be more or less suited to the use of the case system. Participant F opined that the type of application (whether it was data-driven or not) would be more important than the area of industry it occupied. Participants G and A agreed explicitly. The only explicit disagreement with this came from participant B, who slightly contradicted his earlier statement by saying that it would not be of a great deal of use in his specialist area, which was geospatial science:

> *The kind of crap I do's just very... geospatial science methods heavy... and I don't know how well this sort of thing could support it.*

After this participants discussed whether certain kinds of mobile platform would be more suitable for the case system than others. They named several platforms as definitely suitable for its use: Android, iOS, Blackberry, WebOS, the Windows family of mobile operating systems and Nokia's old Series 60. None of the participants noted any mobile platforms that were capable of running applications that were unsuitable for this approach.

Participant D suggested also that it would be a suitable platform for building apps for modern game consoles, which also have a user interface in which the user interacts with one presentation (in the sense of the word introduced in chapter 4) at once. Participant F, whose professional domain was in smart televisions and multi-screen systems based around smart televisions, continued this discussion, noting that it would also be suitable for use in smart televisions. He also stated that in his opinion, the issue that was holding the AppMaps tool back from being more universally applicable was not the case system but the automaton. He was of the opinion that if petri nets (see Peterson, 1977), or some other formal structure that allowed the system as a whole to be in more than one state at once, were in use instead then the system would be of use in multi-screen environments.

7.3.4 Discussion

The workshop created a brief and somewhat artificial situation. Developers were not working in-depth on applications of their own; they had little time to grow

used to the tool or the case system; and they were only assembling pre-made components, instead of writing their own source code.

With these limitations in mind, however, it was possible to draw out some brief and cautious answers to the questions posed in chapter 4 with regards to the workshop, and also some prognostications about the usefulness of the tool and specifically about its acceptance by developers in the longer study.

Below are presented answers to the questions initially posed in chapter 4.

7.3.4.1 Effectiveness

Were developers happy with what they had produced? Developers during the workshop said that they were happy with the quality of the parts of the interfaces that they had produced using the case system.

Did their solutions resemble a sane model answer? The application maps produced by developers differed, as outlined above. When only the assignment of cases to edges is examined, however, corresponding edges in the different application maps bear the same case.

Where did they feel the tool fitted into the problem of plasticity? Participants used, unprompted, terminology very similar to that of the Arch model to describe the problem of plasticity as it applied to their professional practice. Specifically, they highlighted the importance of the physical layer to making usable mobile applications. As noted above, participants outlined a number of important areas that were purely physical that they felt had a huge impact on the quality of the interfaces that they produced. More specifically, these areas were not areas that they felt were considered in *any* of the cross-platform tools they had used, rather than just being a problem with AppMaps. There was a general agreement that the AppMaps case system provided useful plasticity at the dialogue layer, but that the perceived quality of the resulting applications might depend more on attention-to-detail at the physical layer, no matter how "good" the dialogue layer was when measured against the user interface guidelines.

Would they find the tool useful in their professional practice? The most compelling evidence that the participants would have found the tool useful in their professional practice came at the very end of the workshop, when the author was approached separately by four of the seven participants asking whether they could get a copy of the software for their own use. Three of these had no particular project in mind, but the fourth wanted to use it in a specific research project.

This adds force to the more nuanced data from the workshop discussion. As outlined above, when this question was used as a discussion prompt, there were some concerns about the usability of the tool in its current state. These concerns, set out above, did not concern the case system, but were to do with issues of the behaviour of the software tool itself.

Apart from participant C who was, as noted above, not a professional developer, the group of participants generally agreed that this would be useful in their professional practice if these issues were addressed.

7.3.4.2 Efficiency

Would this tool make the development process more efficient, and if so, which parts of it? Participants felt that the tool would increase their efficiency, especially when the focus of the application development process was on novel internals, and they wanted to have a conventional user interface. Participants did not directly talk about what development phases the tool would be useful in, but instead talked about changing the way that they structured the process of user interface development, allowing them to design the dialogue in a separate process.

Would the case system of the tool be worth learning for the expected efficiency gains? Here, participant C's status as a self-professed non-expert developer was illuminating. Her commentary that she felt that she understood the case system of the tool to a useful extent after having used it for an afternoon, and that it was no worse than any other technical language backed up assertions by other participants that they would be glad to put in the effort to learn the tool because it provided useful assistance.

7.3.4.3 Expressiveness

Are there platforms or form factors to which this tool is not applicable? Participants were of the opinion that the tool was very widely applicable among mobile devices, and suggested several form factors outside of the mobile sphere where the tool could be used. It was suggested that the case system itself was not subject to the same limitations, however, and that a system based on a different kind of dialogue model might be applicable to a wider range of platforms.

Are there kinds of applications to which this tool is not applicable? Participants agreed that the software was useful for "a very interesting subset" (participant F) of applications, those based around data being manipulated or browsed. This was identified by them as an extremely prevalent kind of application.

Are there industries to which this tool is not applicable? Participant B was the only person who expressed doubts about the usefulness of the AppMaps model to them on the grounds of application area. No other participant identified industrial areas in which the tool or model would not apply.

7.4 Independent development studies

The workshop study was followed by a longer study in which developers had more freedom, and chose for themselves what task to attack using the tool. In this study participants were given several days with the tool.

To prepare for this study, a number of the user interface issues that participants brought up in the workshop were addressed: the behaviour of the text fields while editing labels was made less idiosyncratic, the quit menu item was made to work correctly and the errors generated by the application runtime when it was given an invalid dialogue machine were made far more communicative. In addition, the author's own work with the tool between the two studies had found two bugs which meant that certain kinds of invalid application map could generate unstable applications that did not deal gracefully.

7.4.1 Method

7.4.1.1 Participants

Seven participants (who will be referred to as participants 1 through 7) were recruited from the author's professional network.

All participants were professional developers who spend a large proportion of their working time doing software development, and all of whom have mobile development as part of their professional practice. All were regularly developing with JavaScript. Five used recent versions of Xcode; three had used the J2ME mobile toolkit.

7.4.1.2 Apparatus

Each participant was sent a copy of the AppMaps software along with a detailed user manual and a set of example projects. They were also sent details of how to obtain a mobile device simulator that would work with the tool. Otherwise, participants used their existing development tools.

The user manual consisted of a tutorial section, a description of the dialogue controller and descriptions of the cases following those laid out in chapter 3.

7.4.1.3 Procedure

The structure of the independent development studies was essentially the same as that of the workshop. Participants began by following the same worked example, then had a question and answer session, then built an application, and then had a debriefing and discussion session during which they provided feedback to the author about their experiences with the tool.

Participants worked remotely, being based wherever they normally did software development work. Each participant was given a one month window in which to spend between two and three days working on an application using the AppMaps tool. With the exception of participant 2, who spent four days on his application, the other participants stayed within the two to three day expectation. Participants 1–3 and 5 did their application development while in the same place as the author, and communication took place face-to-face. Other participants communicated with the author by telephone or by various internet communication media. For participants who could be co-located with the author, audio was recorded of meetings, and any other questions raised were noted in writing. Where participants were not co-located with the author, voice calls were recorded, emails were archived and text chat sessions were logged.

Each participant began by working through the same satellite navigation exercise as in the workshop, but working from the manual rather than from a presentation. The manual took them through the same steps of building and styling an application map and plasticising it using case. It then added an extra step in which the developers added a new view to the project to walk them through using the JavaScript API.

After this, each participant had a discussion period with the same emphasis as the first discussion period in the workshop with the author, in which any areas of the tool's operation that were immediately unclear could be discussed. Participants 1–3 and 5, who were co-located with the author, asked their questions face-to-face. Participant 6's question and answer session also took place face-to-face. Participants 4 and 7 requested that their question and answer sessions take place over Internet Relay Chat ("IRC"). Audio was recorded of face-to-face meetings, and logs were kept of the IRC sessions.

After this, developers began working on their own application. The author was available to act as technical support. The author did not interfere in the architecture or design of the system that participants built.

The development phase ended when either the participant felt that the application was complete or when they were unable to dedicate any more time to the application. At this point, a second discussion took place between the participant and the author, analogous to the second discussion period in the workshop.

Similarly to the workshop, the discussion hinged around the three Es, and each question asked in chapter 4 was used as a conversation prompt. The applications that the participants had built were returned to the author, at which point the author counted the software components involved in plasticity for each application and examined the parts of the application where case had been used in order to evaluate those parts of the application against the user interface guidelines of the respective platforms. Again, for participants 1–3, 5 and 6, this took the form of a face to face meeting where audio was recorded. For participants 4 and 7, this too occurred over IRC, which was logged.

The participants' remarks and comments were categorised under the 3 Es according to their definitions in chapter 4. In the case of remarks that touched upon two areas, they were categorised under both. The discussion session at the end used the three Es as discussion prompts, and this made the categorisation of participants' remarks from this session trivial, as they all remained on topic through the discussion period. No points raised by participants were discarded.

Ethics approval was granted for this study.

7.4.2 Results

The applications that developers built are summarised here: further details, along with application maps and screenshots can be found in appendix C.

Participant 1 built an application for internal use in a publishing company that he runs in addition to his practice as a developer. This application let him browse data relating to the workflow involved in publishing books, and look at what stages each book was occupying in this workflow. He considered the application to be complete when he submitted it.

Participant 1 had reservations about the quality of the user interface produced, although he was happy with the parts of the interface that the case system had generated. He felt that case had made a major impact on the efficiency of his development process, especially in making the jump between "concept and prototype". He also felt that the level of plasticity that the tool provided was sufficient for what he needed. He felt that the case system had been well worth learning given the efficiency gains that he had appreciated, but felt that more cases would be useful. Specifically, he mentioned a case for instrument or tool selections (compare the hypothetical use of the instrumental in chapter 3).

Participant 2 built an application to display and browse the catalogue of his book collection. He considered the application to be complete when he submitted it.

Participant 2 was happy with the quality of the user interface produced. He felt that the case system had made development more efficient and more predictable. He found the case system very intuitive and did not feel that much learning had been required. He found that the plasticity provided by the case system was sufficient for his application.

Participant 3 built a prototype of a social media application that let users leave textual notes at geographical locations. He considered the application as a working prototype when he submitted it, but the system as a whole and the application were unfinished.

Participant 3 was not happy with the quality of the user interface produced, due to physical factors in the interface. He was happy with the parts of the interface that case had generated. He had found that case had made the development process more efficient largely because it had helped him explore alternative versions of the application more quickly. He felt that the case system was certainly be worth learning given this efficiency gain.

Participant 4 built a to-do list application. She fell victim to bugs in the AppMaps tool, and so did not complete her application, but submitted it as unfinished anyway.

She felt that case had been useful in separating ways in which the user could interact with the application and that the cases represented these ways well. She did, however, say that there could be more cases supplied by the application, mentioning again a tool or instrument case. She said that if the tool had worked correctly then case would have provided a significant efficiency gain for her, and that it would probably have provided the degree of plasticity that she needed.

Participant 5 built a prototype of an application to allow users to browse and contribute to a crowdsourced dataset about the accessibility of venues. The application was a working prototype when he submitted it.

He was happy with the quality of the user interface that he had created, saying that his application had little in the way of complex cosmetic needs and that case had adapted the dialogue well. He mentioned that case had provided a "huge practical benefit" in terms of efficiency, and that the level of plasticity was sufficient for his application. He felt that the case system was intuitive and that he didn't feel that he had really had to learn it at all.

Participant 6 built a flashcard game for language learners. He submitted the application as working but needing more polishing.

He was happy with the quality of the user interface for his own use, but said that the physical user interface looked unprofessional and he would not want to release it for other people. He did, however, think that the dialogue adaptations that the tool made using the case system were appropriate. He felt that he had had an efficiency gain from being able to consider both the phone and the tablet at once during development, rather than having to switch between periods of considering only one platform at once. He thought this was enough benefit to justify learning the case system.

Participant 7 designed an online shopping website for parts for Volkswagen Corrado cars. He submitted the site as unfinished.

He felt that he was on a path to creating a high-quality user interface using the tool, although with some reservations about the physical quality of the underlying Sencha Touch toolkit. He felt that the case system had saved efficiency by factoring out common patterns that he would otherwise have had to write boilerplate code for and by allowing exploration and refactoring of the user interface rapidly.

7.4.3 Discussion

From the software produced by these participants and their ideas and opinions expressed during discussion (which are presented in detail in appendix C, it is now possible to synthesise answers to the questions posed in chapter 4.

7.4.3.1 Effectiveness

Were developers happy with what they had produced? Participants 1, 3, 4 and 6 were not happy with the quality of their user interfaces as a whole. Their reasons for this were centred around the physical user interface: participant 4 especially singled out the visual quality of the controls and the quality of the scrolling animations, and participant 1 complained about the typography. Participant 3 felt that the whole interface was rendered badly because of the dependence on the web browser. These concerns do not fall under the responsibility of case.

When the question was reframed to take case into account and to examine only the areas of the interface that case touched, developers' responses were different: the dialogue structure that case produced was perceived as good quality (with the exception of participant 4, who said that "if case had been implemented correctly" it would have been good quality) and the physical adaptation of that bit of the interface was good quality (except, again, for

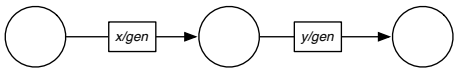


Figure 7.5: Two adjacent genitive edges

participant 4). For these four participants, however, this was not enough to propel the interface as a whole up to being a good one.

Participants 2, 5 and 7 were happy with their interfaces; they all stated that their problem spaces were fairly formulaic.

There was no connection between whether participants were happy with their interface and whether they identified the job they did as 'design' or 'development'. Participants 1, 6 and 7 all identified their job as 'application design or web design'; the others identified themselves as developers. This suggests that this division between people who were happy and not happy with their interfaces does not straightforwardly emerge from a designer/developer distinction.

Did case contribute to the application's conforming to the human interface guidelines for the desired platforms and form factors? Not all the resulting applications had dialogue components that corresponded to the user interface guidelines for tablets. Specifically, participant 4's to-do list application did not comply, and nor did the early version of participant 2's book catalogue that had a genitive search edge.

A glance at why this occurred, however, shows that the iniquity sits on the shoulders of the implementation of case, rather than the model. Specifically, these cases all happened when two genitive edges were next to one another, as shown in figure 7.5.

In this situation, according to the definition of the cases and how they map to user interface elements presented in chapter 5, one should end up with a split view arrangement. The way that this view works is that if a genitive is selected in the left hand panel, and there is another genitive selection to come, then the contents of the left hand panel are replaced with the second genitive view. This can be seen clearly in Apple's Mail application for the iPad (see figure 5.3): first, the user selects a mailbox. The mailbox list is then replaced in the left hand view with the list of messages: and when the user selects one of those, only then does the right hand panel change.

However, the graph grammar rule for genitives presented in chapter 5 and which was used in the implementation of the AppMaps tool does not actually implement this user interface pattern correctly. Instead, when the rules

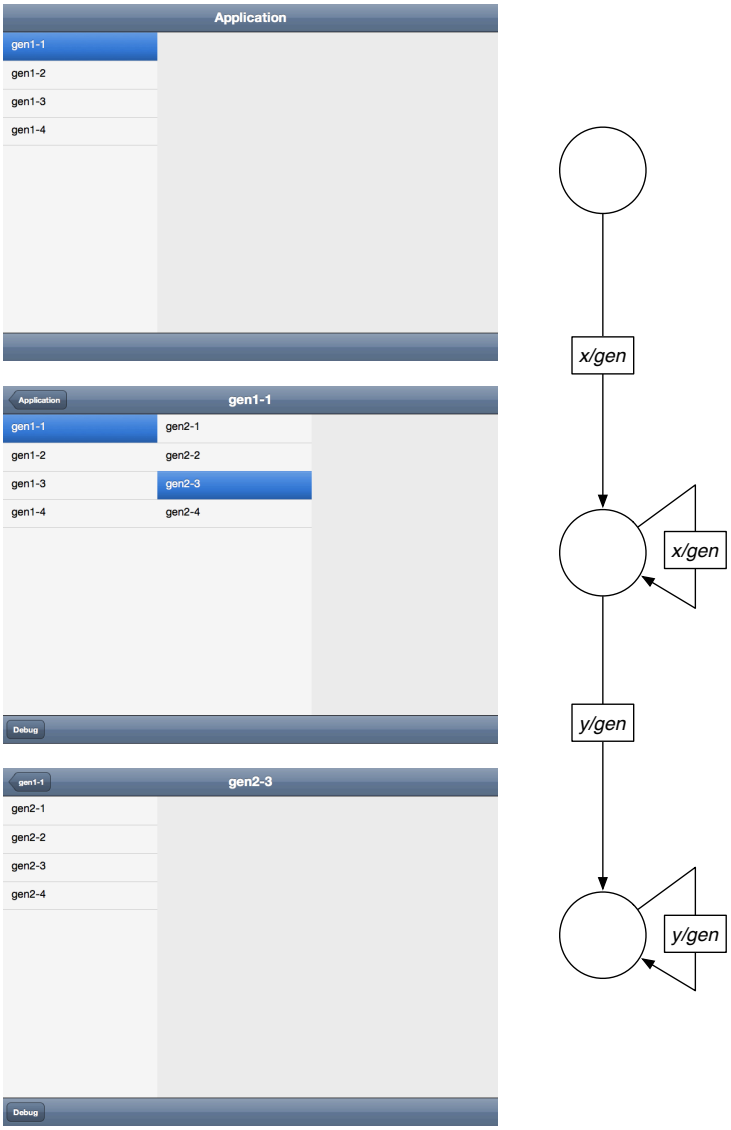


Figure 7.6: Incorrect rendering of adjacent genitives

are actually applied it presents an interface like that in figure 7.6: the middle state ends up with *two* genitive edges coming out of it, which are styled according to the stylesheet: and the middle state ends up as a three-way split view, which does not adhere to the guidelines.

According to the case system, however this subgraph with two genitives one after the other is perfectly permissible and has a well-defined outcome which does conform to the user interface guidelines both on Android and iOS. If this nuance of the case system had been correctly implemented, then all the

parts of the application touched by case would correctly adhere to the user interface guidelines for Android and iOS .

In applications that did not use the pattern of two consecutive genitives (all the final applications but for participant 4's) the use of the genitive produced a user interface that obeyed the rules for both Android and iOS. Likewise, all uses of the dative produced user interfaces that complied to the Android and iOS human interface guidelines.

7.4.3.2 Efficiency

How many software components are involved in effecting plasticity? In all of the applications that the developers built, all plasticity was managed by the cases on the edges of the map and, in some cases, in an application-specific stylesheet. No platform-specific stylesheets were used, and no individual views performed their own plasticity. In other words, the number of software components involved in effecting plasticity in all these applications was constant, regardless of the number of views that needed to adapt.

If a hypothetical tool were used where views had to provide their own plasticity if they needed to adapt between platforms, then the number of software components that were needed to effect plasticity would at least be equal to the number of views that needed to effect plasticity, and thus would scale linearly with that figure.

It is worth noting, however, that in these simple applications the number of edges that bear either a genitive or a dative marker is quite small; and so the apparent large disparity between being constant and being linear in the number of views providing plasticity may not be as large in practice.

Does the use of case provide a tangible benefit in terms of time at any specific point in the development process? One recurring theme in the feedback from the developers is that the amount of time taken to build a user interface of the quality that they got was markedly lower with the AppMaps system than they expected.

The exact point at the development process at which this was perceived to take place varied; and it is also worth noting that the developers applied case to their application map at different points. Participants performed the three stages of building the application map, applying case to it, and implementing the business logic of the application in a number of different ways, despite the documentation only demonstrating a single order (in which the cases were applied after the map was designed).

One strong agreement between participants (explicitly being mentioned by participants 1, 3, 4, 5 and 7) was at the point where the case system provided

the most benefit: it provided it early in the development process, where the connections between "concept and prototype" (participant 1) were being made, and when the "big picture" (participant 4) of the application was being decided on, or a prototype was being refined "in a client situation".

Would the case system of the tool be worth learning for the efficiency gains experienced? All the participants except participant 4 indicated that the case system was worth learning for the efficiency gains that they had experienced. Participant 4 was of the opinion that if the implementation of case were to be corrected then it would be worth learning. When prompted further, none of the participants had any reservations about making this statement. Participant 5 said that he hadn't really felt like he'd had to learn much, more a process of "getting used to" how the cases behaved.

Further evidence that the developers learned the case system rather than just copying the documentation can be found in the uses of case that were not mentioned anywhere in the documentation: the use of searches as genitive by participants 2 and 5, and the use of datives for destinations other than social media by participants 1 and 4. Participant 6 had an application of a very different form from the wine shop or satellite navigation examples, but still used the dative and genitive appropriately in this situation.

7.4.3.3 Expressiveness

Does case work well with the platforms the developer used? None of the developers identified any mobile platforms where the case system would not apply.

Does case work well in the kinds of application the developer created? All of the developers studied produced data-driven applications. All of these applications used genitive edges, and most used dative edges. Participants 2 and 4 reinforced that case had worked well for them because they were working with data-driven applications, and participant 6 noted that theirs only worked because they were building an application that involved two distinct data sets, or two views on the same data at different levels of detail: and that if they had been trying to build an even simpler application, case would have been entirely useless. This is logical, given the definition of case as being about the relationships between objects; if there are no relationships between objects (just, for example, between objects and actions) then case cannot come into its own. Participant 7 noted that the functionality needed for e-commerce applications, of browsing, filtering and sharing, was an ideal match for this case system.

Does case work well in the industries that the developer was working in? None of the developers felt that case was domain-specific or industry-specific. They were in unanimous agreement that the kind of application was more important than the industry, and that one could reasonably expect to find data-centred applications in all industries.

7.5 Reflection upon method

The results above are shaped, both in their essence and in their mode of presentation, by the method used to generate them. The first and most obvious way they are so shaped is that they are nearly purely qualitative. A second way that the method shaped the results is that the results of both the workshop and the self-directed development study are based on fairly short acquaintance with the tool. More subtle problems with the tool or the underlying model would perhaps have emerged with time and familiarity.

However, two points suggest that the data is representative. First, it is broadly in agreement with the results of the case studies, adding detail to the picture gained from the case studies rather than contradicting them. This, however, could be a function of similar assumptions being made in each study. Secondly, and perhaps more tellingly, participants themselves felt that the data recorded,l and the way that this data was presented, was representative of their experience using the tool. This also makes it unlikely that the categorisation of discussion into the 3 Es missed any major issues or aspects of the experience of using the tool.

The methods for these two studies could be reused to perform a similar evaluation, with some care. The 3 Es are not independent of one another, and while for the purposes of these studies they proved to be independent enough to be discussed separately, it is not certain that they would be sufficiently independent for the same to hold true for evaluation of some other tool or framework. For example, if a developer could not produce a good user interface at all (effectiveness), it is also meaningless to speak of their efficiency.

7.6 Summary and conclusion

In this chapter, the part of the evaluation of the AppMaps case system that relied on other developers was described. This consisted of two studies, one short workshop study in which developers gave their feedback in fairly general terms based on a prescribed application-building exercise, and one longer one in which developers built an application of their choice and gave feedback on the specific development process they had experienced.

In the next chapter, the results from this chapter and the two preceding chapters will be synthesised into answers to the research questions that were originally given in chapter 3.

Chapter 8
Discussion

8.1 Introduction

Chapters 6 and 7 presented the results of the evaluation exercises used to work out if the tool that was produced in chapter 5 did useful things for developers, and through that whether the case system that was embodied in that tool was of use to developers. This chapter takes those results and discusses them to draw out common themes between the evaluation exercises.

First, it returns to the research questions that were presented in chapter 3, and synthesises answers to them from the results of the evaluation exercises. From these results, meaningful answers can be given to all of the research questions.

Second, having bounded the applicability of case through the research questions, the chapter looks at what the scope of case is: under what circumstances the positive answers to the research questions are valid, and for what platforms and kinds of application. Third, it reflects upon the methodology that was followed to reach these answers, and considers whether these methods could be reused elsewhere and whether they could be used to reproduce the results given here. Lastly, it considers the limitations of the research, laying out several points upon which the results shed no light, and upon which it would be interesting to have light shed.

8.2 Are the categories delineated by case relevant to user interfaces?

When this question was introduced in chapter 3, three areas were identified as being critical to getting a positive answer. These three areas were firstly that the case system should be intuitively usable to developers, secondly that the case system as used in the tool must behave like a case system, and thirdly that case must be a living phenomenon to users of the case system, capable of being used in unexpected ways. In chapter 4 it was noted that an answer to the first needed to emerge from the expressiveness information gathered from other developers, an answer to the

second needed to emerge from the design of the case system itself, and an answer to the third must emerge from the case studies and the longer development study.

The workshop and self-directed development study provide good evidence for the first area. Not a single developer in either study said that the case system had been unintuitive in use or unpredictable. When participants were asked whether it was intuitive they all said that it was; and some (such as participant 5) had felt there was nothing to learn at all. Participant 2 specifically noted that the case system had contributed to the tool's predictability; and participant 4, whose project fell foul of software defects, had a sufficiently strong and accurate mental model of the case system that she could confidently (and correctly) assert that it was the software that was at fault rather than her understanding the case system.

Developers did have one major obstacle to overcome to find their intuition of the system, however. A theme that strongly emerged from the workshop was that the terminology (genitive, dative, oblique) was obscure and opaque. This was especially clearly articulated by participant C in the workshop, but was entirely absent as a concern during the longer development workshop.

It must be said that developers are not alone in finding the terminology surrounding case difficult. The vocabulary used to describe it—not just in this thesis, but wherever it is studied—is both opaque and often of ancient origin. According to Butt (2006), the words for the individual cases are first attested in the Greek grammar of Dionysios Thrax in the second century BC. These names were then translated—or, in the case of "accusative", *mis*translated—into Latin by Remmius Palaemon in the first century AD. The names currently used are anglicised versions of these Latin names. It is, therefore, not surprising that developers found these names confusing, because the meanings of the names are not transparent. This failure of terminological clarity does not, however, invalidate the evidence that developers found the case categories themselves intuitive. Participant C also articulated this clearly, saying that once she had got past the names, the categories themselves made sense.

The second area, whether the case system sufficiently resembled the case system of a natural language, was addressed by the construction of the tool. The case system of the tool was based on Blake's case hierarchy, with the omission of the top two cases, which are purely linguistic. This follows the normal analysis (presented, for example, on page 33 of Blake, 2001). There are theorists who would disagree with this, especially with regards to the accusative case: localist case grammars would count the accusative as having a semantic meaning (such as in Anderson, 2009), and cognitive grammar frameworks hold that everything in language has semantics (Luraghi, 2009). Therefore, given the construction alone, it is very likely but not certain that the constructed case system mirrors natural case systems closely enough to be worthy of the name.

However, there was an unexpected and extremely suggestive piece of evidence for the case system as embodied in the tool being closely related to case in language. This evidence came from participants 1 and 4 in the self-directed development study. These participants did not speak any language that had a complicated case system—participant 1 speaks English and German and participant 4 is monolingual English. Neither had either of them come into contact with academic writing on case. However, both of these participants suggested that the case system could do with being extended, and both suggested the same extension, that of a case for a tool.

This is suggestive because an instrumental case is also on the case hierarchy, with the fundamental 'case function' of encoding the tool by which an action is performed (Blake, 2001, section 5.7). It is not the next case down in the hierarchy after the genitive—the locative, which encodes physical position, fills that position—but it is worth noting that neither participants 1 nor 4 built applications that had any geospatial or positioning component at all, and so a locative would not have been relevant to them anyway.

Two participants is not a large sample, even within the fourteen developers studied. However, there are not many cases in the case hierarchy. The top four specific cases were already accounted for by the implementation of the tool. That two participants who are unaware of the structure of case systems should suggest one of the remaining two cases on the case hierarchy as a natural extension to the case system of the tool seems to fit the model too neatly to dismiss immediately as noise. This is an area that would repay further study.

Evidence for the third area, that case was a living phenomenon to developers, emerges from both the case studies and the longer workshop studies. The genitive was originally demonstrated and explained to participants in terms of a list of items or categories that refines either a second set of items or a details view (as in the satellite navigation example given to workshop participants in chapter 7). In the Speakers Associates case study, spurred on by the commissioning person, this was extended with a genitive search. In the longer self-directed development study, this genitive search was mirrored in participant 2's book catalogue and participant 5's map-based application. Participant 5 considered putting the map in the genitive, because of its partitive meaning, but fell foul of the same software defect as participant 4. This demonstrates that the genitive was certainly not being used as a frozen idiom involving a list of items.

The dative was used by fewer participants but with more variety. It was originally demonstrated to participants for social media destinations; but its uses multiplied, always with the basic meaning of 'destination' or 'beneficiary'. The case studies contributed examples of its use for physical devices (EVENT2) and for departments of a company (Agritechnik). The self-directed development study also

contributed an example of its use for addressing parts of a company (participant 1) and logical destinations within a filing system inside the application itself (participant 4). This demonstrates that the dative, too, was not being used simply as a way of generating user interfaces for social media destinations.

In summary, then, developers perceived the case system of AppMaps as being intuitively usable and easy to learn, where learning was required at all. This case system bore a significant structural similarity to natural language case systems, and feedback from some participants suggests strongly that they were thinking about the case system unconsciously in these terms. This case system was not being used as a set of frozen idioms, but as something capable of being used by developers in the service of the problem to hand, with a set of meanings that were capable of being applied outside of expected situations. Therefore, it is reasonable to answer this question positively: the categories delineated by case *do* seem to be relevant to user interfaces, at least in the context of mobile cross-platform development.

8.3 Can a tool be built that embodies case for building plastic user interfaces?

When this question was raised in chapter 3, two areas were given that needed to be considered in any answer. The first was that the tool had to do something useful with the cases that were provided by the developer, and the second was that the parts of the applications generated by this process that had been touched by case had to obey the user interface guidelines for the platform. In chapter 4 it was stated that the information to support the first area needed to come from the act of building the tool itself (as presented in chapter 5), and the information from the second emerges from both the case studies and the longer studies.

In chapter 5, the user interfaces that should emerge from the use of the genitive and dative cases were defined. First, they were defined in terms of the user interface guidelines; and second, they were defined in terms of graph grammar rules over a simple dialogue state machine (called "the application map"). If the second had been a perfect implementation of the first, then this answer could have been extremely short. However, the second was not a perfect implementation of the first.

The implementation of the genitive on the two tablet platforms had a major defect in it. On tablets, the genitive was supposed to be implemented as a split view type display. When two genitives were placed next to one another in the application map, then the results did not obey the user interface guidelines and in fact were quite unpredictable (this bug affected the Agritechnik application in chapter 6 and participants 2 and 4 in the self-directed study in chapter 7).

If this defect could not be resolved, then case could not be said to be implementable. Fortunately, however, it can be resolved. If the graph grammar rule in figure 5.21 is amended so that the second edge only matches non-genitive edges then the behaviour no longer occurs. If a chain of subsequent genitives occurs in the application map, only the last one will have the extra duplicated genitive edge added to it. This means that no state in the application map will have two different genitive edges going from it, which prevents the situation where multiple left-hand panels would be created on tablets and a guideline-violating three-way split view would be generated.

The implementation of the dative on all platforms had a smaller shortcoming. Two possible realisations in terms of concrete user interface components for the dative were noted: one for when the user was choosing a destination from a set of destinations, one for when the user was entering a freeform address or phone number. The former was to be rendered as a button in a popover, the latter was to be rendered as a normal freeform field.

In the AppMaps tool as given to participants, only the former was implemented. As before, if this problem is not amenable to being resolved, then case could not be said to be implementable. Again, however, it is resolvable. The reason why the latter realisation was not implemented in AppMaps is because the decision as to which realisation to use in the application needed to be done based on the actual component in use. If the edge marked with the dative were a button or a set of buttons, then the first realisation needed to be used; if the edge marked with the dative were a text entry control, then the second needed to be used. However, because of the way the type system of JavaScript works and the way that Sencha Touch's classes work, this information was not available at application build time. In the AppMaps system, the decisions about what concrete realisation to use for cases were made at build time. This meant that no decision could be made based on the kind of component that was in use.

To resolve this defect would require the use of a toolkit that made that information available at design-time. Any object-oriented toolkit where each kind of component is represented by a distinct, declaratively-defined class would, for example, provide this information. This could then be used to make the decision as to which realisation to use.

By contrast, the evidence that the parts of the resulting applications that are generated by case obey the user interface guidelines for the platform is clear. None of the participants—except for those who fell foul of the genitive software defect described above—produced applications that violated the user interface guidelines of the platforms they were developing for. Nor was it any effort on the present author's part to adhere to the user interface guidelines during the creation of the case study applications. This suggests that the fairly simple implementation of

cases in terms of user interface components that the AppMaps tool embodies is actually sound, and the two categorisations line up well.

In summary, the AppMaps tool is an example of a tool that implements the genitive and dative cases in software. The implementations it provides are inadequate, but are inadequate in ways that can be addressed. If these inadequacies were to be addressed then its implementation of cases would produce parts of applications that obeyed user interface guidelines. This question, too, warrants a positive answer: cases, at least a genitive and a dative, have been implemented in software.

8.4 Can case as embodied in this tool be used to build useful applications?

Chapter 3 stated that a positive answer to this question needed to emerge from applications built using the tool for other people. Chapter 4 stated that this could be addressed by examination of the case studies and the self-directed development study in terms of the three Es. Two of the participants in the longer study have created applications that they consider as 'complete' and which they use in their everyday lives: these were participant 2's book catalogue and participant 6's quiz application. However, since the other participants were also attempting to create useful applications, then their feedback should also be considered here.

8.4.1 Effectiveness

In both the case studies and the longer development study, two questions were considered under the heading of effectiveness. The first of these was to do with the commissioning user's or developer's perception of the quality of the interface, and the second was to do with whether the parts of the interface that case touched adhered to the user interface guidelines of the platforms that the developer was targetting. The second of these has been examined in section 8.3 above, but the former has not yet been discussed.

There is strong agreement between the results of the case studies and the results of the two developers from the self-directed development study who produced useful applications (in the strict sense noted above) about both the quality of the parts of the interface that case touched and in terms of the limitations of case. In both, the quality of the parts of the interface that case touched was perceived as good. In chapter 6, the three commissioning users all stated that they were content with the quality of these parts of the interface in their respective applications, with the commissioning user of the Speakers Associates application specifically pointing out that these aspects were formulaic. Likewise, both par-

ticipants 2 and 6 considered that the parts of their user interface that case had touched to be of good quality. This was also the general opinion of the other participants, who produced applications that they did not go on to use.

However, the parts of the interface that case had touched were by no means the whole of the interface, and the parts of the interface that case did not touch outnumbered the parts that it did by a significant margin. The evaluation for effectiveness in chapter 6 noted that, by volume, case accounted for very few of the guidelines in the user interface guidelines, and that the specific warnings that Google make in their guidelines about cross-platform interfaces do not touch any of the same areas of the interface that case does. In the self-directed development study, participant 2 thought that the interface he had produced was entirely sufficient "for [his] purposes" and had no opinion beyond the application's usefulness to he himself. Participant 6, on the other hand, thought about potential distribution of the application, and reached the conclusion that the user interface was not in a state where it could sensibly be distributed, because other aspects of the interface other than those that case touched were not of sufficiently high quality, notably the parts that the Arch model would refer to as the "physical" layer of the interface.

In this, participant 6 was not alone: five of the seven participants in the longer study had concerns about the quality of the physical elements of the interface that they had produced with the tool. That the dialogue component of the interface might have been slightly imperfect might not have stopped them shipping their application: but the deficiencies in the physical aspects of the interface did prevent them from thinking that their application was good enough to release to other people. Thus, at least as far as participants' perceptions of the quality of their interface went, the physical layer was granted more of an importance than its position as an outlier on the Arch might make it seem like it warrants.

AppMaps delegated the physical layer of the interface to the underlying Sencha Touch toolkit. It provided a stylesheet that let the user set options in that toolkit, but the actual physical aspects of the interface were up to the developer and to the underlying toolkit. The other plastic user interface tools follow a similar approach of farming the physical layer out. UIML2, as defined by Phanouriou (2000), has a whole set of transformations: but still, the final physical decisions are delegated to the developer (who specifies the styles of components), the renderer (which implements UIML in terms of an underlying toolkit) and the toolkit that the renderer finally targets. As noted in chapter 2, the mechanism by which ARTstudio generates the final interface is unclear: but the reference framework it exemplifies (see Thevenin et al., 2003) specifies that all stages of the interface should be under the developer's control, and from the images provided of the tool in use it seems to use the inbuilt components for MacOS Classic and PalmOS.

TERESA and MARIAE/MARIA both eventually hand off the problems of physical display to other toolkits, with varying degrees of control over this process for the developer. DiaGen is designed for web applications and is bound to HTML. It is also worth noting here that, according to the feedback from the workshop participants, even the industrial tools, which focus single-mindedly on physical and logical adaptation (see chapter 2 and the data in appendix A), do not get the physical adaptation of interfaces even nearly correct. AppMaps gets the physical layer wrong: but it does so in good company: it is worth noting that when Paternò et al. (2011) evaluated the MARIAE tool in a workshop situation, the area that was "most problematic" (p. 1821) was the quality of the final user interface.

AppMaps was, however, first and foremost a tool to evaluate the usefulness of case. There are limits in how applicable case is, and one of those limits is that there are many factors that go into a physical user interface—such as those identified by the workshop participants earlier in this thesis—that it does not capture. However, the fact that the underlying Sencha Touch toolkit generated poor-quality physical user interfaces does not undermine the effectiveness of case within its limits, any more than a poor-quality renderer would undermine the fundamental ideas of UIML, or a bad HTML renderer would undermine the usefulness of DiaGen. Note also that developers were successful in commenting on the usefulness of case as distinct from the rest of the tool: in their feedback they were firm that their answers attached to the underlying case system, rather than to the tool itself.

8.4.2 Efficiency

In both the case studies and the longer development study two questions were considered under the heading of efficiency. The first of these was to do with how many software components were necessary to effect plasticity, and the second was to do with whether developers had felt that the process was more efficient, and which points in the development process case helped most with. A third was added for the longer study, where developers were asked whether they thought case had been worth learning.

In the case studies, the only software components of each application that were involved in effecting plasticity were the dialogue controller and the stylesheet. This remained the case even when an unexpected platform was added to the requirements part of the way through the development process, as in the Agritechnik case study. Towards the end of chapter 6, this was compared with hypothetical processes of development using pure MVC and an MVCD architecture that did not include dialogue plasticity. In the former, the number of software components that effected plasticity would at have been linear in the number of views that needed to adapt to different platforms; in the latter, linear in the number

of platforms addressed. The only difference between the latter of these and the model as implemented in AppMaps was that AppMaps used the case system to provide dialogue plasticity. Since the number of software components required to effect plasticity in AppMaps remained constant even when the number of platforms increased, this gain can be ascribed to case. It is unlikely to be purely a quirk of AppMaps' implementation of case, as it was shown above that AppMaps' implementation of the case system follows the structure of said case system very closely, and introduces no extra meanings or nuances to the cases themselves.

This was backed up by the two participants' applications in the longer development study. In both of these, similarly, the number of software components required to effect plasticity was constant. Applying the same hypothetical development scenarios to the two participants' efforts gives similar answers, and argues that case is the agent that permits plasticity to be effected with a constant number of software components involved.

The phases of development at which case is most useful, according to both the case studies and the results from the self-directed development study, are the exploratory ones where different structures are being tried out. Exactly when during the development process this takes place varies between participants, however.

In the case studies in chapter 6, the major perceived efficiency gain by the author was in exploring alternatives. This stood out most strikingly in the exploration of alternative realisations of search in the Speakers Associates application, and the explorations of alternative Windows Phone interfaces for the Agritechnik application.

Participants in the longer study expressed similar ideas. Participant 1 talked about case making it easy to move from concept to prototype. Participant 3 said that he had approached the exercise with a firm idea about how his application ought to work, and that he thought this had prevented him from using the case system to its full potential. Overall, as discussed in chapter 7, they thought it would be useful when the "big picture" (per participant 4) was being decided upon. Also, as noted in chapter 7, with the exception of one participant, the participants were unanimous in their feeling that, given the efficiency gains they had experienced, it had been worth learning case.

Statements about efficiency do emerge from evaluations of other plastic user interface development systems, but not in sufficient detail to permit immediate comparison. While the evaluation of MARIAE (in Paternò et al., 2011) seems mostly to have been concentrated on which parts of the development environment worked well for users, some indication that developers thought that it would increase their efficiency do appear. In this evaluation there was a strong response from developers that the process of generating a task structure from annotations applied a web service "can help in speeding up the development." (sic; p. 1820)

However, since this part of the evaluation applies to the process of generating the task structure and not the process of actually making this task structure appear on multiple devices, the question of a direct comparison of efficiency between MARIAE and AppMaps remains open. Similarly, the account of the evaluation of the DiaGen tool (Book et al., 2006) mentions a very impressive reduction in the amount of code needed to implement a given application feature (user registration), but does not provide any detail on precisely where this efficiency gain came from, and acknowledges that this is a very coarse measurement that does not bear straightforwardly on larger applications. Therefore, the question of direct comparison here also remains open.

8.4.3 Expressiveness

The evaluations of expressiveness in the case studies and longer studies were performed by looking at three areas. The first was whether case adequately covered the platforms required, the second was whether case was more useful for some kinds of applications than others, and the third was whether case was more useful for some industries or domains of application than others.

As stated in chapter 5, in its original form the AppMaps tool catered to Apple's iOS and Google's Android platforms. For the purposes of the Agritechnik case study, the tool was extended to cover Windows Phone: however, this support was not sufficiently mature to give to other developers for the self-directed development study.

In the case studies, the case system met the expectations of the commissioning users with regards to what platforms and form factors it was implemented for; and likewise, participants were unanimous in saying that the tool catered for the platforms and form factors they wanted to develop for. This is possibly because Android and iOS, and phones and tablets, are the most visible in the ecosystem and have the most market share. For the commissioning users, and for the external developers, this result is a good thing; but in the context of this thesis it is disappointing. There have been no failures; so there is no firm evidence for what the limits of case's applicability in platform terms is.

The case studies and longer development study are more useful in finding out whether case is more useful for some kinds of applications than others. Three kinds of applications were built for the case studies: the first application was primarily a communication application, the third application was purely to do with browsing a data set, and the second application had a mixture of both functions. The communication application did not use case very effectively; in the data-driven application it was extremely useful; and in the application that mixed the two functions, it was of most use in the data-driven portion. All the developers in

the longer study built applications that manipulated or browsed data sets. Again, however, a strong theme that emerged in their comments was that the case-based approach would only work with data-driven applications. This follows to a great extent from the definition of case itself: if case as implemented in the tool is about the relationships between distinct objects in the application it will mostly be applicable where there are distinct objects that the user is manipulating for there to be relationships between.

When compared to the other tools surveyed in chapter 2, AppMaps is definitely the poor relation in terms of expressiveness in terms of number of platforms covered. Even ARTStudio, which only covers two platforms (Thevenin, 2002), covers one desktop and one mobile platform; TERESA covers an impressive number of platforms, including desktop, mobile and voice-activated (Paternò et al., 2008). However, it must be remembered that AppMaps was specifically trying to solve a problem for mobile developers, and was intended only to address these platforms. It cannot be concluded from this that case itself is only applicable in these contexts.

8.4.4 Conclusion

The combination of the case studies and the self-directed development study gives good indications that case *can* indeed be used to build useful applications, being of most use for those which are data-driven in some way. It makes a contribution to having a good quality interface by plasticising the dialogue layer, but this is not in itself enough to make the resulting interfaces high-quality. It does, however, provide a meaningful gain in efficiency compared to the tools that developers currently use, especially in the early, exploratory stages of the development process.

8.5 Can case as embodied in this tool be used by other developers?

A broad indication that case as embodied in the AppMaps tool is of use to other developers can be drawn from the fact that, at the time of writing, there is a small and nascent group of developers using AppMaps consisting of a subset of the participants in the longer development study. The tool can be downloaded from `http://appmaps.co.uk/appmaps-bin.zip`. Only time will tell whether this community survives. This broad indication is not enough to generate an answer to this question, however.

In chapter 3 it was stated that any evidence of the usefulness of case for other developers needed to come out of the mouths of other developers. Chapter 4 refined this to state that an answer should emerge from the workshop and self-

directed development study, evaluated in terms of the three Es. On the whole, the feedback from the workshop and the feedback from the longer study agreed very well.

8.5.1 Effectiveness

Like the self-directed study participants, the workshop participants were happy with the quality of the parts of the interface that case had generated, but had doubts about how large the problem of dialogue adaptation was compared to the problem of plasticity as a whole. The emphasis in their doubts was also the same as those of the longer study participants: they considered that the physical elements of the interface had a major influence on the quality of the resulting interface, that it was this physical adaptation that they put a lot of effort into when building cross-platform mobile applications, and it was this physical adaptation that was not supported well by cross-platform tooling in general. Even given this shortcoming, however, the workshop participants still thought that the use of case would find a place in their professional practice.

8.5.2 Efficiency

The self-directed study participants felt efficiency gains in the specific projects they had attempted to build, and the count of software components involved in plasticity in their projects backed this feeling up. The workshop participants talked with a wider scope about their professional practice in general. They were strongly of the opinion that they would find an efficiency benefit from the use of the tool in projects they undertook, and that it would be worth the effort to learn the tool. Therefore, it is reasonable to conclude that case could provide an efficiency benefit not just to the author but to developers in general.

8.5.3 Expressiveness

Much like the evaluation of expressiveness for the self-directed development study, the evaluation of expressiveness in the workshop study concentrated on three areas: whether the tool and case system were applicable to the platforms developers wanted to target, whether the tool and case system were applicable to useful kinds of applications; and whether the tool and case system were applicable to useful industries. Again, the results of the workshop study closely mirrored the results of the self-directed development study.

The results in the first area broadened the picture of which platforms a case-based approach might be appropriate, but did not help in setting the boundaries of case's usefulness. While the self-directed development studies showed that

developers thought that case would be useful on the mainstream mobile platforms, the workshop study expanded this to potentially being applicable to things that are not mobile platforms at all, such as smart televisions and games consoles. They did not, however, venture an opinion on where it would not be useful, so the limits of case's applicability across platforms remains a murky question.

The results of the second area reinforced and added context to the results from the self-directed development study. All of the applications that the self-directed development study generated were data-driven applications, and the developers considered that this was the kind of application that case was most applicable to. The participants in the workshop study agreed with this but contextualised it by noting that these data-driven applications are extremely common, and forms a very "interesting subset" of mobile applications.

8.5.4 Conclusions

The case system was readily used by other developers in both the workshop and the self-directed development study. The benefits that other developers derived from the platform were very similar to those that the author derived during the case studies.

The developers who participated in the two studies gained much the same efficiency benefit from using case that appeared in the case studies. Case was useful to them during exploratory parts of interface-building, and they derived the same quantified efficiency benefit as was demonstrated in the case studies.

Both the case studies and the workshop provided some indication that case would be useful to other developers across all the mainstream mobile platforms, but neither set limits to its usefulness. Therefore, the question of where case stops functioning in terms of platforms remains unanswered. However, both these studies also showed that case would be most useful in data-driven applications and would be less useful outside of this. This, too, agreed with the results from the case studies.

In conclusion, therefore, other developers derived much the same benefits from the case system as the author.

8.6 The scope of case

It would not be reasonable to expect case as embodied in AppMaps to be useful in all circumstances. From the above answers, it is possible to extract some adumbrations of the limits of case.

First, case only works for certain kinds of applications. Both the case studies and the feedback from the participants in the studies demonstrate that case is of

most use when there is data involved, and there are relationships between items from those data sets and other items, or between items in those data sets and actions the user can perform. It is of little use for communication applications or applications where there are no discrete data items. It seems likely that this can be said to be a limitation of case itself, because case is defined in terms of the relationships between things. If there are no discrete things for there to be relationships between, then case will accordingly be less useful.

Second, according to developers who used it, the AppMaps case system is well-suited to a range of platforms. The platforms that they said would be suitable for it shared a notable characteristic with mobile platforms: that the user was interacting with only one presentation, or one presentation per application, at once. This is probably not a limitation imposed by case itself, but a limitation imposed by the dialogue component. The state machine that forms this component can only be in one state at a time. This conclusion is indicated by participant F's noting that if the dialogue model could be somehow in multiple states at once then the AppMaps system would be useful for display ecology applications; it is indicated by the example in chapter 3 outlining how a case system might apply to an application on a desktop computer; and it is indicated by the fact that there is nothing in the structure of case that requires the user only to be interacting with one presentation at once.

Third, it is most useful at early stages of the development process. This was evident from both the case studies and the studies with other developers. Later in the development process, the decisions that case facilitates seem to be overridden by other concerns. This, too, seems likely to be a limitation of case itself. The categories delineated with case are to do with the ways that selections that the user makes relate to one another, and this is quite a high-level structure. As the changes being made to the application get smaller and smaller and more towards polishing the application ready for release, the less likely it is that changes on these scales will happen.

8.7 Comment on methodology

The methodology used, as outlined in chapter 4 and with the details given in chapters 6 and 7 captured a wealth of useful information about developers' use of the tools, including, reassuringly, areas that were originally intended as separate evaluation criteria. The methodology as a whole did not rely on specific circumstances: there was nothing in it specific to that set of participants or the AppMaps tool and as such it could be easily reused either to attempt to reproduce the results in this thesis or to evaluate other user interface tools.

8.7.1 The three Es

The three Es captured all the information that was expected, and some that was not. In the original plans for the workshop study, the developer's ability to use the tool and the hunting of software defects in the tool before the longer study were to be enquired about separately as an independent part of the workshop. In practice, however, these plans were pre-empted by participants who naturally wove these discussions into the discussion of effectiveness and whether the tool would have a place in their professional practice. The separateness of the description of the scope of case does not contradict this: while it was not evaluated within the three Es, this is not because it did not fit within any of them, but because it drew data from all of them and synthesised it into a whole.

If used elsewhere, the three Es would need to be used with some care, because they are not entirely orthogonal. In this thesis, this did not become problematic: but, for example, the criteria of effectiveness and expressiveness are tied together, because how good the quality of any given user interface might be (effectiveness) might well depend on what kind of interface it was or what platform it was on (expressiveness). Likewise, the criteria of effectiveness and efficiency are somewhat tied together, because if the tool produced a lower quality user interface that the developer felt was redeemable, they might spend time polishing the interface and thus lower their apparent productivity.

With that caveat borne in mind, here is no reason why the 3 Es are specific to case, or even to plastic interfaces in general. The similar, but non-identical, set of criteria in ISO 9241-11 (comprising effectiveness, efficiency and satisfaction) are aimed at evaluation of usability in general. These three criteria concentrate on the user's experience in using a system, while the three Es concentrate also on the developer's ability to produce output using the system. However, they are similar enough that they back up a case that the three Es would be generalisable outside the AppMaps context.

8.7.2 The contexts of evaluation

None of the contexts in which evaluation was done was controversial, as shown in chapter 4. In the context of this thesis, they demonstrated their worth both individually and in combination. They provided three views, at three levels of detail, of developers' interactions with the AppMaps tool and with the case system embodied therein. The two most detailed provided quite deep insights into the usefulness of case and the tool, and the workshop study provided context that was not elicited by either of the other studies. These three approaches could easily be reused elsewhere in a similar combination, so long as criteria could be found that provided points of comparison between the results from each of the contexts.

Again, there is nothing specific to AppMaps about the combination of the three Es and the three contexts: and a very similar methodology combining both could doubtless be used to evaluate other tools or frameworks and possibly to compare them with AppMaps or case.

However, there was a very visible weakness of the two longer evaluation methods, the case studies and the self-directed development studies. They both required a significant investment of time on the parts of participants. In the first, the author's time with the commissioning people inside the companies for whom the applications were being built was limited by those people's other commitments. In the second, many of the participants were taking time away from paid commissions to take part in the study. Participants gave very generously of their time—they consistently gave more time than was asked of them—but even so, their time was a limiting factor. This made it a somewhat risky approach: there was a danger of wasting participants' time and goodwill and getting few results. Participants were already hard to recruit for studies of this length.

It is for the reason of this shortage of participants' time that, for example, only one of the case studies draws a direct comparison between an application built using AppMaps and a similar application that does not. It is also for this reason that the self-directed development study did not provide comparisons between AppMaps-built applications and similar applications not built using AppMaps: participants' time simply did not stretch far enough to make this practicable. This absence of comparisons is a limitation of the results presented within this thesis, and forms a fitting point to begin discussing the limitations of the research done.

8.8 The design of the tool and the understanding of case

There are two kinds of understanding that participants might have had about case. The first kind is a conscious, intellectual knowledge about case: the second kind is an unconscious knowledge. The first kind corresponds to a linguist or grammarian's understanding of a language: the second to a speaker's. The design of AppMaps might potentially have influenced either of these kinds of understanding, or indeed have helped them to interact by bringing an awareness of an unconscious process to awareness.

The design of the AppMaps tool certainly imparted the first kind of knowledge to most of the participants. One of the participants in the longer study knew some German, and thus was familiar with the concept of a genitive case; another was learning Polish, and thus was familiar with both the concept of a genitive and the concept of a dative case.

Traditionally, case has been understood and taught through a Greco-Roman lens. As noted above, the very terminology that case distinctions are expressed in

is Greco-Roman. The first contact with case that many people have had has been in compulsory Latin lessons. Whether learning about case from a system like AppMaps produces similar kinds of understanding to learning about case in the traditional way is an interesting question, but unfortunately there is no evidence about this that can be drawn from the studies presented here.

On the second kind of knowledge, there is at least an indication that can be drawn from participants' comments. It was noted above that two participants suggested an extension to the case heirarchy that was consistent with Blake's case heirarchy, and also that participants both in the case studies and in the self-directed development study used case in ways that were not part of the original design or documentation of the tool, but were consistent with the meanings of the cases.

This is meagre evidence, but such as it is it suggests that participants' unconscious understanding of the case system, or of whatever underlies it, was not entirely created by their exposure to AppMaps.

8.9 Limitations of the research

8.9.1 The absence of comparisons

One striking omission in the preceding chapters has been the absence of comparisons. Case has been demonstrated as useful, but has not been demonstrated to be *more* useful than anything else except the tools that developers currently use. There are three major areas of comparison that could fruitfully have been pursued. The first of these is a comparison of case with other tools and frameworks for plastic user interfaces, the second is a comparison of case with other sets of semantic roles, and the third is a direct user-centred comparison of applications resulting from a case-based design process with applications built using other methods.

8.9.1.1 Case vs. other plastic UI models and tools

Aside from the case study for Speakers Associates, where a comparison was made between the application built with AppMaps and a similar application built with unadorned Sencha Touch, no direct comparison was made between a case-based approach to plasticity and other approaches.

Taking the research tools summarised in chapter 2 first, this absence is due to two factors. The first, touching on the case studies, is the author's complete failure to produce applications with any of the research tools that were comparable to the applications built in the case studies. The second is that developers were not aware of these tools or able to use them. This was a circumstance that was put forward as likely in chapter 2 and was confirmed by the demographic questionnaires given

out before the workshop and before the self-directed development study. In these questionnaires, not a single developer had used or even heard of any of the research tools mentioned in chapter 2, so a direct comparison was out of the question.

Taking the industrial tools second, here the situation is slightly more complicated. There was implicit comparison between the case-based process and the process involved with other tools both in the workshop and in the longer development study. Developers drew commonalities and differences between AppMaps and other cross-platform development tools, especially when comparing their effectiveness. Developers were also asked whether they were more efficient because of case, which also involves a comparison between the case-based approach and their normal approaches. But, apart from the Speakers Associates case study, there was no direct comparison. In both the case studies and the longer development study, this shortcoming arose from the length of time for which the author had access to participants. In the latter case, especially, attempts to recruit developers for a study in which they built the same application twice, once with their normal cross-platform tooling and once using AppMaps, met with a stunning lack of enthusiasm.

Without these comparisons it is hard to say exactly where case fits in the context of the wider plastic user interface literature and the tooling that this literature has produced. It is also not possible to say whether the same advantages that case offers could be provided by different tools in different ways: while this thesis has argued that case is an effective solution to the problem it sets out to address, it cannot claim that case is a uniquely powerful solution to this problem.

8.9.1.2 Case vs. other sets of roles

In chapter 3, the theories of cognitive grammar and localist case grammar were adduced as theories that provided a basis for an idea of case based on an underlying set of semantic roles. Other sets of roles, which occupy a variety of areas in linguistic theory, have also been proposed, and some of those are likely to be amenable to being implemented in a similar way to the way AppMaps implements cases.

One immediate candidate would come from the very oldest systematic treatment of case, in the Aṣṭādhyāyī of Pāṇini (available in English as Vasu, 1891). The Aṣṭādhyāyī is a grammar of Sanskrit that was written about 600 BC and is still engaged with in computational linguistics about the Indic languages (for example, see Subbanna and Varakhedi, 2009 and Goyal et al., 2009). This set of categories is fit for implementation because, like the case system that it analyses, it is both small and well-defined.

Other candidates that likewise are plausibly implementable are the "Deep Cases" of Fillmore (1968) and Starosta's Lexicase (see Starosta and Nomura, 1986) as well as the two mentioned in chapter 3. These theories are all compared and analysed by Blake (2001).

Without these comparisons, it is possible to say that case is useful to developers, but it is not possible to say that case is any *more* useful than any of the other semantic sets of categories would be. Nor, without such a comparison, is it possible to contribute to the debate on how case relates to these categories.

8.9.1.3 Applications vs. Applications

In chapter 4, the user's interests were said to be represented by the user interface guidelines. This meant that there was no evaluation of the resulting applications based on user testing. This meant that there was no comparison from the user's point of view between applications built using the case-based approach and applications that were not, and it is impossible to say whether the user could tell the difference between the two. As noted above, of the case studies, only the Speakers Associates app was built both using traditional cross-platform tools and using AppMaps. Even if this had not been the case, and the other two had also had versions not built with case, and a user-based comparison had been done, this would not necessarily have proved very much. To complete the picture, the applications that developers had built would also need to have been built in two forms and tested and, as noted above, it proved impossible to get participants for a development study in which the same application needed to be built twice.

Without this comparison, information about the quality of the interfaces produced with case relies on the expert opinions of developers and the people who commissioned the applications, and compliance to the user interface guidelines, which are far blunter instruments than one might wish.

8.9.2 The absence of formal methods

In chapter 5, the dialogue controller was specified as a formal construct based on a state machine. The cases, too, are partially defined formally, through the graph grammar that produces the final dialogue state machine for different platforms. Even given this definition, however, the subsequent evaluation of the tool had no formal component whatsoever.

There is a connection between formal properties of a state machine for a system and the usability properties of that system (Thimbleby, 2007). Care must be taken, however, in comparing the use of state machines in AppMaps with other work, because the state machine in AppMaps does not model the state of the entire system, but only the state of the dialogue component of the user interface.

This is a result of using an MVC-based architecture, where much of the state about the system as a whole is stored in the system's model objects. This does mean, however, that the conclusions drawn from a formal analysis of the state machine would probably be more limited than they would be for a system where the state machine held the state of the entire system.

The obvious place where formal evaluation might have played a role for AppMaps is in the expressiveness criterion. This criterion was about what kinds of user interfaces can be created using the tool. What kinds of graph can be created by the graph grammar might have been a fruitful thing to investigate under this criterion.

8.9.3 The absence of quantitative measurements

No fine-grained quantitative measurements emerged from the evaluation of AppMaps. Some numbers did emerge: the efficiency evaluation asked about how many software components were involved in effecting plasticity and the expressiveness criterion asked how many platforms could be addressed by the case-based approach. Both of these, however, were low-resolution indications rather than high-resolution measurements.

For the efficiency criterion, certainly, higher precision measurements could have been taken, either using lines of code (as Book et al., 2006, did when evaluating DiaGen) or using function points. However, as mentioned in chapter 4, measuring developer productivity is famously difficult, and although the numbers might have been more precise, whether they were measuring anything useful would have been more in doubt. The effectiveness criterion is more doubtful, although user testing of the generated applications had it been done might have generated some quantitative data. The expressiveness criterion was not amenable to direct measurement at all, because it asked questions about a very open-ended set of platforms and types of application. One approach that could have been followed was that of the evaluation of MARIAE, where data was gathered from the workshop and from the study involving external developers by means of a premade questionnaire. However, a reliance upon that might have prevented the unexpected and interesting remarks made by participants from being made at all or from being given importance in the results.

The fact that the information gathered from participants was qualitative does not devalue or invalidate it. However, some direct measurements would have provided another angle from which the results could have been seen.

8.9.4 Summary and conclusion

The results of the evaluation exercises, which were given in previous chapters, provides a basis for answering the research questions that were posed near the be-

ginning of this thesis. The categories delineated by case do seem to be relevant to user interfaces: the case system of AppMaps is intuitively graspable by developers with little effort, the case system of AppMaps does behave like a case system, and case is a living categorisation to developers, which is capable of being used in the ways that they want. A positive answer to this question is backed up by some unexpected evidence from two of the participants in the longer development study, who suggested extensions of the case system that are in line with the structures of case systems in general. These participants did not know about the structures of case systems.

A tool can be built that embodies case for building plastic user interfaces. The tool was the AppMaps tool as used for evaluation throughout this thesis. It implemented plasticity based on the cases in its case system, rather than just letting the user manipulate them, and the parts of the applications that it generated that were based on case stood up well to being analysed in the light of the user interface guidelines.

Case, as embodied in the AppMaps tool, can be used to build useful applications. Two of the participants in the longer study are, at the time of writing this chapter, using the applications that they built in their everyday lives, which certainly argues for their usefulness. The parts of the applications that case produced were of high quality, but in other areas AppMaps suffered from the same shortcomings as other cross-platform tools. Case did provide an efficiency benefit in the building of useful applications, and its expressiveness was sufficient to this purpose.

Case, as embodied in the AppMaps tool, can be used by other developers. Its usefulness is not a peculiar fantasy of the author's. Other developers took well to the tool and used it without much difficulty to produce non-trivial applications.

Case as used for user interfaces is, of course, limited in scope. It is of most use to data-driven applications where there is a set of discrete *things* being manipulated by the user. It is less useful in situations where there is only very simple interrelations between the things that the user is using and other things in the application or actions that they can take.

The methodology that was used to reach these results produced good information and, if use with some care, could be reused elsewhere, either to re-evaluate this tool with other participants or to evaluate other tools either for comparison with this one or as stand-alone evaluations.

Finally, there are limitations to the research described in this thesis. There were insufficient comparisons: this means that while this thesis demonstrates the usefulness of case, it cannot argue that case is any more or less useful than any other approach. Specifically, it would have been useful to have comparisons between AppMaps and other tools for creating plastic user interfaces, be they from

the industrial or research communities; between the AppMaps case system and proposed 'underlying' sets of semantic roles that have been suggested underlie language; and between applications built using a case-based approach and similar applications built using a more traditional approach. In addition, there was an absence of formal methods used to evaluate the AppMaps tool, which meant that the formal aspects of the dialogue controller as specified in chapter 5 have not, perhaps, been evaluated as thoroughly as they might have been. There was also no quantitative measurement performed.

The next chapter draws this thesis to a close by summarising the contributions it makes, and by suggesting future work.

Chapter 9
Conclusion

9.1 Introduction

This dissertation has argued that case is a useful conceptual model to mobile application developers who need to build plastic mobile applications that react well to changes in platform and device form factor. To conclude, this chapter summarises the contributions that it made and discusses work that could be done in future.

9.2 Contributions

9.2.1 A comparison between industrial and research tools

Chapter 2 presented a comparison between the industrial tools for plasticity for mobile applications and research tools. This comparison was couched in the terms generated by the research community. That this comparison was useful and successful demonstrates that the tools are solving similar enough problems to be talked about in the same terms. Frameworks such as the Arch model (Bass et al., 1992) and the Plastic User Interface Snowflake (Thevenin et al., 2003) can be used to analyse these tools to get an insight into the problems that developers are having and how they are attempting to solve them.

This evidence leads to an extension Coutaz's (2010) statement: not only HCI researchers, but working industrial developers are "address[ing] user interface plasticity from different starting points, depending on their "credo": at the toolkit level by those who advocate "hard core development" and fine grained control of user interfaces, at the infrastructure level with the development of dedicated middleware by those who strive for generic computational substrates, to task level modeling by those who believe in the top-down development of user interfaces." (p.1) Industrial developers are doing this in a disjoint way with no real overriding model or principle, much like the research approach that Coutaz deplores. However, they are inhabiting the same problem space, and are plastic user interface tools within the definitions of the plastic user interface literature.

That they do inhabit this problem space is relevant to work that follows two different approaches to plastic user interfaces. The first approach, which this thesis adopted, was to try to extend the existing practice of developers. This was done so that developers could rapidly pick up the the tool that was developed and use it with enough expertise to provide feedback. To follow this approach one must know about the current practices of developers, because otherwise one cannot accommodate and expand on those practices. The tool presented in this thesis adopted and extended not only current practice but current software: it used Sencha Touch, the cross-platform mobile development toolkit that was, at the time of writing, the most popular among developers. Equally, however, a tool could be built from scratch but using the models and mental frameworks of the development tools should this be a better fit for the specific research question being examined.

A second approach that can be taken is to investigate the usefulness of a specific approach to writing software to the problem of plastic user interfaces. This, for example, is the approach taken by the model-driven tools summarised in chapter 2: they are part of a research programme to investigate and demonstrate the usefulness of model-driven engineering in this context (see Coutaz, 2010). In this context, the current practices of industrial developers are not necessarily of a great deal of interest: if one is looking at the long term development of the software engineering discipline, paradigms and ways of working have come and gone, and developers' practices have changed. JavaScript and MVC will eventually go the way of the keypunch and the autocoder. If one is looking towards the future and investigating how things *might* be done, it is appropriate to disregard more of the details of how things are *currently* done.

However, even if one is taking this approach, the industrial tools are not non-entities. They do not only provide information on how industrial developers are solving their problems with plastic and multi-target interfaces, but they provide information on what developers think these problems are and how they prioritise the parts of these problems. As mentioned above and demonstrated in chapter 2, the industrial tools *are* amenable to being analysed using the frameworks that the research community have developed, and this suggests that the shape of the general problem space that is presented for example by Thevenin et al. (2003) is sound. However, the industrial tools are not randomly scattered within this problem space. They have certain qualities in common that reflect the preoccupations of the developers that use them. These preoccupations embedded into the software provide another view on the problem of plasticity, and should be taken into account.

9.2.2 A demonstration of the usefulness of case

The research questions that are central to this thesis centre around the notion of case and its usefulness, and the main story of the thesis is a demonstration that case is useful and assists mobile application developers in building plastic user interfaces for multiple mobile software platforms and form factors. The requirement on case was to plasticise the dialogue components of mobile applications in such a way that it provided some noticeable benefit to the developer of the application over and above developing different versions of the application by hand. Chapter 8 drew the strands of evaluation together to argue that case is useful in this context, and can be used to build non-trivial and useful applications by developers in general.

The results show that the parts of the interface produced by case are of high quality. The apparent connection between the cases and the user interface guidelines of Apple's and Google's mobile operating systems was backed up by evidence gained from developers' experiences, and the tool that embodied case managed to use this connection to produce adequate quality user interface components. The results also show that developers got the kind of efficiency gain one would expect from correctly-functioning dialogue plasticity: they did not have to design special cases for individual platforms or form factors. The results also show that the approach is applicable to a large and useful class of mobile applications, although by no means all mobile applications. Further, these advantages are not peculiarities of the author's development practice: other developers who used the tool to create applications gained much the same advantages as the author, and workshop participants backed this up by saying that these were advantages they would expect to get in their professional practice if they used the tool.

This primarily provides information to people who are looking to engineer plastic user interfaces. The case-based approach is different from other approaches that have been tried. It draws the seeds of the plasticity it provides from language and from linguistic ways of making word order flexible, rather than from an examination of software. Also, the developer annotates their dialogue model with semantic categories (such as "destination" and "category") rather than with "syntactic" constraints on ordering (such as "enabling", "choice" and so forth as used in ConcurTaskTrees; see Paternò, 2003).

Secondarily, and if used with care, this information may be of use to linguists investigating what case is and what the mechanisms behind it are. The methodology used in this thesis is one appropriate for evaluating software tools, rather than a linguistic one, and so work would need to be done to bridge the gap between these two methodological stances. If this work can be done, however, the information in this thesis could provide a starting point for an argument within

linguistics that some set of roles that underlies case has a wider existence than just within language *sensu strictu*.

9.2.3 An implementation of case

One of the research questions presented in chapter 3 was about whether case could be implemented in a tool. As part of the evaluation of the usefulness of case, this was done in a tool called "AppMaps". The AppMaps tool implemented case using graph grammars and integrates case with an existing MVC architecture.

A description of the way this tool is implemented in chapter 5. Case could be implemented to a useful degree using graph grammar rules that were not particularly complicated, and this work could form a basis for future investigations of the scope of case.

In addition, the software tool itself is now available for use, licensed under an open source license. A description of a software tool may not be enough for many purposes. Certainly, for the investigations in chapter 2, the present author was very glad that in many cases the tools themselves—both from the research community and the industrial community—were available. If the tool itself is available, edge cases can be explored, the claims made about the tools can be validated, and the tool can be used in ways that the original authors perhaps did not intend.

At time of writing, the number of users of this tool is very low. Even if, however, this number does not grow, the software will still be available for continued study or for critique in the future.

9.2.4 The scope of case

Case is not, of course, a plastic UI panacea. It has its limits, and therefore it has a scope in which it is useful. Outside of this scope it is irrelevant or possibly even counterproductive to attempt to apply it.

While some aspects of the scope of case remain obscure, this thesis has contributed information about much of it. It is definitely useful for the three major mobile operating systems and the two major form factors of device in common use at the time of writing. Developers indicated that it is likely to be of use for many other kinds of platform. It is definitely useful for data-driven applications, in which the user is manipulating datasets that are not entirely defined at the time the app is designed. Developers noted that this constituted a major part of the application ecosystem on mobile devices. It is of much less use outside of this area: applications where there are no objects represented that can have interrelations will not benefit much from a mechanism to classify such interrelations, and this is ultimately what case is.

Except in that certain industries have a tendency to build certain kinds of applications (for example, telephony companies might tend to build non-dataset-driven communication applications), there is no evidence that this approach is tied to a certain industry or application domain. The meanings that case, as used in this thesis, delineates are independent of the meanings of the objects these cases are attached to.

9.2.5 The MVCD architecture family

Chapter 2 noted that MVC architectures were common among the industrial tools surveyed. MVC is, it must be reiterated, not a single architecture but a family of loosely-connected architectures that share key concepts and share some of the ways that the software components that make up the application are connected to one another: specifically, they share the concept of a view, which displays information, a model, which stores information, and a controller, which mediates between the two and implements business logic.

In some recent tools, another pattern has emerged inside the MVC pattern. In these architectures, in addition to the three normal kinds of MVC component, there is an explicit dialogue component. These architectures separate task structure and navigation structure from the business logic much more than traditional MVC architectures. In the industrial tool landscape, the most obvious example of this is Apple's Storyboards approach (Apple, Inc., 2012b). In this approach the developer designs the dialogue between the user and the machine in a separate software component. Other industrial examples can be found in MVC frameworks built on top of NetBeans' J2ME Mobile Toolkit (Motocoder, 2006; Keegan et al., 2006) and the Facebook SproutCore framework (Sarnacki et al., 2012): a research example that very clearly shows the pattern can be found in the Dialogue Flow Notation and its application (Book and Gruhn, 2004; Book et al., 2006).

In chapter 5, this thesis drew out commonalities between these tools and outlined the nature of the newly-christened MVCD architecture family, then used this concept to help to structure the AppMaps tool. An MVCD architecture is an MVC architecture with a separate dialogue controller where the dialogue structure of the application is defined by that dialogue component, and the business logic of the application is in the controllers. The MVCD architecture family is thus as broadly-defined as the MVC architecture family, as most MVC architectures would be amenable to being converted into an MVCD architecture.

Because of its broadness of definition, the term "MVCD" is comparable to the term "MVC": it specifies a broad approach, rather than forming a detailed definition of how the software architecture it applies to is put together. It will therefore be of more use as a way of grouping architectures together for comparison, or as a

guideline for the creation of software architectures, than as a way of pinning down the behaviour of individual software architectures.

9.2.6 A generalisable evaluation methodology

The methods that this thesis used to evaluate AppMaps and the case system that it contains are not restricted to this use. Both the combination of the three Es and the three contexts and the three contexts alone are potentially reusable for evaluation of other tools. As noted in chapter 4, the most similar methodology used to evaluate any of the tools in chapter 2 was that used to evaluate MARIAE, which used a similar threefold division (Paternò et al., 2011)..

The three contexts of evaluation provide a very useful cross-section of the development process. The broadest perspective is given by the workshop, where developers gave feedback on the impact of case on their development process in general. A middle perspective is given by the self-directed development study, where multiple developers give information about the application of the tool to a number of projects, but without being expert users of the tool. The finest detail but narrowest scope is given by the case studies, where a few projects are done and described in detail by the author of the tool as an expert developer. These three provide information all the way from nebulous potential future applications of the tool to the very specifics of using the tool to build applications for customers. This provides a wider set of perspectives than the three that the evaluation for MARIAE used, because in that evaluation external developers were given a set task to solve, rather than being left to pursue their own agendas and projects.

The three contexts were not, however, evaluated in this order, because there is a second useful property of the three: by doing the case studies first, then the workshop, then the self-directed development study, many of the software defects in the tool had already been found and solved by the time of the self-directed development study. The most glaring software defects were found during the initial testing of the tool and the case studies. After this, the workshop provided more information about software defects and, especially, interface defects. Thus, by the time that external developers were given the software, they had a far better experience, and thus could provide better information, than if the contexts of evaluation had been performed in order of level of detail.

The combination of the three Es and the three contexts made sure that the information gained from the three contexts were comparable. While the evaluation of MARIAE used a similar three contexts, at least in the published evaluation (Paternò et al., 2011) no detailed comparisons or contrasts were made between the three contexts. The account of that evaluation did not give the exact structures of the questionnaires in use, so it is difficult to say whether such a comparison could

fruitfully have been done given the data that they had. Similarly, in the UIML2 evaluation of Phanouriou (2000), the workshop study is used to answer a specific question about the accessibility of the tool, rather than being part of a framework of comparisons. As argued in chapter 8, the three Es captured a great deal of useful information about the usefulness of the case system in a way that again, was not tightly coupled to anything structural about the tool or the case system. This combination of three Es and three contexts is thus more powerful than the methods used to evaluate other tools and is potentially useful elsewhere.

9.3 Further work

In chapter 8, some limitations of the research done were listed. These limitations lend themselves well to becoming seeds for future work. In addition, the research presented in this thesis could potentially form the seed of a broader contribution to the theoretical debate about the nature of case.

9.3.1 The usefulness of sets of semantic roles

As noted in both chapters 3 and 8, there have been a number of proposed sets of universal semantic roles that underlie both case systems and other parts of language, such as cognitive grammar (Luraghi, 2009), localist case grammar (Anderson, 1987), Lexicase (Starosta and Nomura, 1986) and Fillmore's (Fillmore, 1968) Case grammar.

The approach AppMaps takes, with an annotated dialogue state machine, is not specific to case: any of these sets of roles could be put into the framework. However, one of the reasons case worked well is that there was a good match between the meanings of the cases and rules in the user interface guidelines (see chapter 3). Therefore, a first measurement of how useful these other sets of semantic roles would be in the context of plastic user interfaces would have to emerge from whether they bear any relationship to the kinds of structures that appear in user interfaces or user interface style guides.

After this, a similar approach could be taken as was taken evaluating AppMaps: using the same methodology and the same kinds of questions should produce sets of results that would be easy to compare with one another.

9.3.2 Case and other frameworks

As also noted in chapter 8, there was no comparison done between case and other frameworks that have been built to deal with the problem of plasticity. Two strands of work could be continued on from this thesis to investigate the con-

nections and contrasts between the case-based approach used in this thesis and the other approaches that have previously been used.

The first is simple comparison, attempting to solve similar problems with case-based approaches and, for example, the model-driven engineering tools whose details were summarised in chapter 2. Again, a similar methodology could be used, examining the effectiveness, efficiency and expressiveness that each of the tools brings to its users, and comparisons could be made.

This is not the only way that case and other approaches could be brought together, however. Recall that in chapter 5, nearly all of the design decisions made during the design of the tool were made to maximise the tool's accessibility to the developers who would be evaluating it. The exception to this rule was the case system, as it was this that was being examined and evaluated. If this restriction is lifted, then many other design decisions could have been made. Case is not strongly linked to the design of the AppMaps tool and does not require that design. This means that there is scope to explore ways that a case-based approach could be linked with, for example, model-driven engineering. There is no obvious reason why case as a model could not fit within these other approaches. If such a reason were found, that would be interesting in its own right.

9.3.3 Comparison of outputs

As noted, again, in chapter 8, there was very little comparison of the outputs of the AppMaps development process to the outputs of either applications that were created with different approaches to plasticity or applications where the different interfaces were entirely hand-created.

A straightforward comparison of this kind could be added to the evaluation found in this thesis, if participants could be found that were amenable. In addition, it could be added to any of the comparisons proposed above, to explore how users perceived the results of the differences in development process, if they did at all.

9.3.4 What actually is case, anyway?

The methodology used to develop and evaluate AppMaps is an appropriate one for the design and evaluation of a tool for building user interfaces. This is fitting, because the research questions that were being asked and the problem space that were being explored were both to do with user interfaces.

However, questions about the origin and nature of case in the general case are linguistic, or at least semiotic, and as such demand a linguistic or semiotic methodology. For this reason, the results given in this thesis do not provide any

decisive data for what case actually is, instead demonstrating that it is useful in a specific context.

However, there are hints here that could be followed up with a more linguistic methodology, especially about how quickly developers picked up the case system, about how easily it was extended to circumstances beyond its initial design, and about the extensions that developers proposed to the case system. Also, the questions raised in chapter 8 about whether the kind of understanding of case gained through using an AppMaps-like tool and through learning a language deserve attention. These are interesting questions, perhaps more so because they emerge from a different discipline from the one in which case usually finds its home: and they open up a new area for dialogue between those who study language and those who study HCI.

References

ACM (2013). Engineering—message from the engineering chairs. http://chi2013.acm.org/communities/engineering. Accessed on 1 August 2015.

Adams, D. Q. (1987). *Essential modern Greek grammar*. Dover Publications, New York.

Adams, D. Q. (1988). *Tocharian historical phonology and morphology*. American Oriental Society.

Adobe Systems, Inc. (2013). Phonegap 2.6.0 API documentation. Available at http://docs.phonegap.com/en/2.6.0/index.html. Accessed on 26 April 2013.

Anderson, J. (1987). Case grammar and the localist hypothesis. In *Concepts of case*, pages 103–121. Gunter Narr Verlag.

Anderson, J. (2009). Case in localist case grammar. In Malchukov, A. and Spencer, A., editors, *The Oxford handbook of case*, pages 121–135. Oxford University Press.

Apple, Inc. (1990). *HyperCard 2.2 reference*, chapter 1, pages 1–68. Apple, Inc.

Apple, Inc. (2012a). *iOS human interface guidelines*. Apple, Inc.

Apple, Inc. (2012b). Storyboards. In *Cocoa application competencies for iOS*. Apple, Inc.

Apple, Inc. (2013). Model-view-controller. In *Cocoa core competencies*. Apple, Inc. Available at https://developer.apple.com/library/mac/\#documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html. Accessed on 1 May 2013.

Apple, Inc. (2014). Submitting your app. In *App distribution guide*. Apple, Inc. Available at https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html\#//apple_ref/doc/uid/TP40012582-CH9-SW1. Accessed on 12 February 2014. Another document, "App Review Guidelines"

also exists but at time of writing is not available to anyone except registered iOS developers.

Aronson, H. (1990). *Georgian: a reading grammar*. Slavica Publishers, Columbus, Ohio.

Austerliz, R. (1989). Uralic languages. In Comrie, B., editor, *The world's major languages*, pages 567–576. Routledge.

Balmer, D. (2011). JoApp documentation: joSubject. Available at `http://joapp.com/docs/\#joSubject`. Accessed on 1 April 2013.

Bancroft, H. H. (1882). *The native races III: Myths and languages*. A. L. Bancroft.

Bansod, A. (2012). Sencha Touch with Windows Phone 8. Blog post. Available at `http://www.sencha.com/blog/sencha-touch-with-windows-phone-8`. Accessed on 20 April 2012.

Barnhart, R. K., editor (1999). *Chambers dictionary of etymology*. Chambers, Edinburgh.

Bass, L., Little, R., Pellegrino, R., Reed, S., Seacord, R., Sheppard, S., and Szczur, M. (1992). The Arch model: Seeheim revisited (version 1.0). The UIMS developers workshop (April 1991). *SIGCHI Bulletin*, 24(1):289–308.

Bergsland, K. (1950). Norwegian research on the language and folklore of the Lapps. part I. language. *Journal of the Anthropological Institute of Great Britain and Ireland*, 80(1):79–88.

Berjon, R., Leithead, T., Navara, E. D., O'Connor, E., Pfeiffer, S., and Hickson, I., editors (2012). *HTML5: A vocabulary and associated APIs for HTML and XHTML*. World Wide Web Consortium. W3C Candidate Recommendation. A formal recommendation document is expected at the end of 2014.

Berti, S., Giulio, M., Paterno, F., and Santoro, C. (2005). TERESA: An environment for designing multi-device interactive services. In *Proceedings of the 4th Italian symposium on human-computer interaction*, pages 40–44.

Bézivin, J. (2004). In search of a basic principle for model driven engineering. *CEPIS Upgrade*, 5(2):21–24.

Binnig, C. and Schmidt, A. (2002). Development of a UIML renderer for different target languages: Experiences and design decisions. In *Computer-aided design of user interfaces III*, pages 267–274. Springer.

Blake, B. J. (1977). *Case marking in Australian languages*. Australian Institute of Aboriginal Studies, Canberra.

Blake, B. J. (1979). *A Kalkatungu grammar*. Department of Linguistics, Research School of Pacific Studies, Australian National University.

Blake, B. J. (1983). Structure and word order in Kalkatungu: the anatomy of a flat language. *Australian Journal of Linguistics*, 3(2):143–175.

Blake, B. J. (2001). *Case*. Cambridge University Press.

Blandford, A. (2013). Engineering works: what is (and is not) engineering for interactive computer systems? In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 285–286. ACM.

Bodnar, A., Corbett, R., and Nekrasovski, D. (2004). Aroma: ambient awareness through olfaction in a messaging application. In *Proceedings of the 6th international conference on multimodal interfaces*, pages 183–190. ACM.

Bonfante, G. and Bonfante, L. (2002). *The Etruscan language: an introduction*. Manchester University Press.

Book, M. and Gruhn, V. (2004). Modeling web-based dialog flows for automatic dialog control. In *Proceedings of the 19th international conference on automated software engineering*, pages 100–109. IEEE, IEEE Press.

Book, M., Gruhn, V., and Lehmann, M. (2006). Automatic dialog mask generation for device-independent web applications. In *Proceedings of the 6th international conference on web engineering*, pages 209–216. ACM.

Booth, D. and Liu, C. K., editors (2007). *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. World Wide Web Consortium. Available online at `http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/`. Accessed 1 May 2013.

Bos, B., Çelik, T., Hickson, I., and Lie, H. W., editors (2011). *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) specification*, chapter 7. World Wide Web Consortium.

Boyer, J. M., editor (2007). *XForms 1.0*. World Wide Web Consortium, third edition edition.

Brewster, S., McGookin, D., and Miller, C. (2006). Olfoto: designing a smell-based interaction. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 653–662. ACM.

Bryant, J. and Jones, M. (2012). Responsive web design. In *Pro HTML5 performance*, pages 37–49. Springer.

Butt, M. (2006). *Theories of case*. Cambridge University Press.

Calvary, G., Coutaz, J., Dâassi, O., Balme, L., and Demeure, A. (2005). Towards a new generation of widgets for supporting software plasticity: the "comet". In *Engineering human-computer interaction and interactive systems*, pages 306–324. Springer.

Cardona, G. (1987). Indo-Aryan languages. In Comrie, B., editor, *The world's major languages*, pages 440–447. Routledge.

Central Statistical Agency - Ethiopia (2007). *The 2007 population and housing census of Ethiopia: Statistical report at national level*. Central Statistical Agency - Ethiopia. Available at `http://www.csa.gov.et/images/documents/pdf_files/regional/CountryLevel.pdf`. Accessed on 26 June 2013.

Clackson, J. P. T. (2008). Latin. In *The ancient languages of Europe*, pages 73–95. Cambridge University Press.

Comrie, B. (1987). Introduction. In *The world's major languages*, pages 1–29. Routledge.

Comrie, B. and Polinsky, M. (1998). The great Daghestanian case hoax. In *Case, Typology and Grammar: In Honor of Barry J. Blake*, pages 95–114. John Benjamins.

Comrie, B., Polinsky, M., and Rajabov, R. (1998). Tsezian languages. Available at `http://scholar.harvard.edu/files/mpolinsky/files/tsezian.98.description.pdf`. Accessed on 20 August 2013.

Cook, S. (2007). *Domain-specific development with Visual Studio DSL tools*. Addison-Wesley, Upper Saddle River, NJ.

Coutaz, J. (2010). User interface plasticity: model driven engineering to the limit! In *Proceedings of the 2nd ACM SIGCHI symposium on engineering interactive computing systems*, pages 1–8. ACM.

Craig, J., editor (2013). *IndieUI 1.0 user context: Contextual information for user interface independence*. World Wide Web Consortium. W3C Editor's Draft 13 May 2013. Available at `https://dvcs.w3.org/hg/IndieUI/raw-file/default/src/indie-ui-context.html`. Accessed on 13 May 2013.

Craig, J. and Cooper, M., editors (2013). *IndieUI: Events 1.0: Events for user interface independence.* World Wide Web Consortium. W3C Working Draft 22 January 2013. Available at `http://www.w3.org/TR/2013/WD-indie-ui-events-20130122/`. Accessed on 1 May 2013.

Daniel, M. and Spencer, A. (2009). The vocative—an outlier case. In Malchukov, A. and Spencer, A., editors, *The Oxford handbook of case*, pages 626–634. Oxford University Press.

Edzard, D. O. (2003). *Sumerian grammar*. Brill.

Einarsson, S. (2000). *Icelandic: Grammar, text and glossary*. Johns Hopkins University Press.

Erich, G., Richard, H., Ralph, J., and John, V. (1995). *Design patterns: elements of reusable object-oriented software*, chapter 5. Addison Wesley Publishing Company.

Estublier, J., Vega, G., and Ionita, A. D. (2005). Composing domain-specific languages for wide-scope software engineering applications. In *Model driven engineering languages and systems*, pages 69–83. Springer.

Favre, J.-M. (2004a). Foundations of model (driven)(reverse) engineering: Models. Episode I: Stories of the fidus papyrus and of the solarus. In *Post-proceedings of Dagsthul seminar on model driven reverse engineering*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI).

Favre, J.-M. (2004b). Towards a basic theory to model model-driven engineering. In *Proceedings of the 3rd workshop in software model engineering*. IEEE.

FeedHenry (2012). Feedhenry documentation: Development 101. Available at `http://docs.feedhenry.com/v2/development.html`. Accessed on 28 April 2013.

Fillmore, C. J. (1968). The case for case. In Bach, E. and Harms, R. T., editors, *Universals in linguistic theory*, pages 1–88. Holt, Rinehart & Winston.

Fortson, B. W. (2009). *Indo-European language and culture: An introduction*. Wiley-Blackwell.

Gabillon, Y., Petit, M., Calvary, G., and Fiorino, H. (2011). Automated planning for user interface composition. In *Proceedings of the 2nd international workshop on semantic models for adaptive interactive systems at IUI 2011*. ACM.

Gamkrelidze, T. V. (1966). A typology of Common Kartvelian. *Language*, 42(1):69–83.

Garcia-Ontiveros, D. (2011). Treasures from the London Library: Knud Leem: an accidental ethnologist in Lapland. *History Today*, 61.

Gençkaya, O. F. (1995). Republic of Turkey. In Flanz, G. H., editor, *Constitutions of the countries of the world*. Oxford University Press.

Goddard, C. and Wierzbicka, A. (2002). *Meaning and universal grammar: Theory and empirical findings*. John Benjamins Publishing.

Goguen, J. (1999). An introduction to algebraic semiotics, with application to user interface design. In *Computation for metaphors, analogy, and agents*, pages 242–291. Springer.

Google, Inc. (2013a). Android: Design patterns. Available at `https://developer.android.com/design/patterns/index.html`. Accessed 31 December 2013.

Google, Inc. (2013b). Android developer documentation: App components: Activities. Available at `http://developer.android.com/guide/components/activities.html`. Accessed 29 April 2013.

Google, Inc. (2013c). Publishing overview. Available at `https://developer.android.com/tools/publishing/publishing_overview.html`. Accessed 26 May 2013.

Goyal, P., Kulkarni, A., and Behera, L. (2009). Computer simulation of Aṣṭādhyāyī: Some insights. In *Sanskrit computational linguistics: Selected and invited papers*, pages 139–161. Springer.

Green, M. (2004). Introduction. In *The Aeneid of Virgil*. Wordsworth Classics.

Gustafson, A. (2011). *Adaptive web design: Crafting rich experiences with progressive enhancement*. Easy Readers, LLC.

Hale, A. and Manandhar, T. (1973). Case and role in Newari. In *Collected papers on Khaling, Kulunge, Darai, Newari, Chitwan Tharu*, pages 79–93. SIL.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

Hargreaves, D. (2006). Review of Newār (Nepāl Bhāsā). *Himalayan Linguistics Review*, 3:1–4.

Henry, S. L. (2013). IndieUI overview. Available at `http://www.w3.org/WAI/intro/indieui`. Accessed 4 May 2013.

Hetzron, R. (1978). The nominal system of Awngi (Southern Agaw). *Bulletin of the School of Oriental and African Studies*, 41(1):121–141.

Holes, C. (2004). *Modern Arabic*. Georgetown University Press.

IBM Corporation (2013). Developing IBM Worklight applications. In *IBM Worklight V5.0.5 information center*. IBM Corporation.

Ifrah, G. (2000). *The universal history of numbers*. John Wiley and Sons, New York. Translated by Bellos, D., Harding, E. F., Wood, S., and Monk, I.

International Organization for Standardization (2011). ISO/IEC 9075:2011: Information technology — Database languages — SQL.

Jagersma, A. H. (2010). *A descriptive grammar of Sumerian*. PhD thesis, Faculty of the Humanities, Leiden University.

Janhunen, J. (2003a). Preface. In *The Mongolic languages*, pages xvi–xxiii. Routledge.

Janhunen, J. (2003b). Written Mongol. In *The Mongolic languages*, pages 30–56. Routledge.

Job, M., editor (2004). *The indigenous languages of the Caucasus*. Caravan Books.

Jones, C. (2000). *Software assessments, benchmarks, and best practices*. Addison-Wesley Longman Publishing Co., Inc.

Jones, P. V. (1998). *Learn ancient Greek*. Duckworth, London.

Jones, S., Voskoglou, C., Vakulenko, M., Measom, V., Constantinou, A., and Kapetanakis, M. (2012). Cross-platform developer tools. Technical report, Vision Mobile.

jQuery Team (2013). *jQuery Mobile 1.3.0 API documentation*. jQuery Foundation. Available at `http://api.jquerymobile.com/`. Accessed on 28 April 2013.

JSR271 Expert Group (2009). *JSR 271: Mobile Information Device Profile for Java™Micro Edition 3.0*, chapter 12, pages 187–614. Motorola, Inc.

Karlsson, F. (1999). *Finnish: an essential grammar*. Routledge.

Keegan, P., Champenois, L., Crawley, G., Hunt, C., Webster, C., Jullion-Ceccarelli, J., Prazak, J., Ryzl, M., Sporar, G., and Wielenga, G. (2006). *NetBeans™IDE field guide: Developing desktop, web, enterprise, and mobile applications*, chapter 14. "Developing Java ME mobile applications". Prentice Hall, second edition edition.

Kennedy, E. C. and Davis, A. R. (1964). *Two centuries of Roman poetry*. Macmillan.

König, C. (2008). *Case in Africa*. Oxford University Press.

König, C. (2009). Case in an African language: Ik - how defective a case can be. In *The Oxford handbook of case*. Oxford University Press.

Krasner, G. E. and Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the Smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49.

Krause, A. (2007). *Foundations of GTK+ development*. Apress.

Langacker, R. W. (1986). An introduction to cognitive grammar. *Cognitive science*, 10(1):1–40.

Lee, C. N. and Thompson, S. A. (1987). Chinese. In Comrie, B., editor, *The world's major languages*, pages 811–833. Routledge.

Lewis, G. (1985). *Turkish grammar*. Clarendon Press.

Lewis, M. P., Simons, G. F., and Fennig, C. D., editors (2013). *Ethnologue: Languages of the world*. SIL International, seventeenth edition edition. Available at `http://www.ethnologue.com`. Accessed 10 August 2014.

Liddell, H. G., Scott, R., Jones, S. H. S., and McKenzie, R. (1940). *A Greek-English Lexicon*. Clarendon Press.

Lipsman, A. (2013). Marketing to the multi-platform majority. Technical report, comScore.

Lipsman, A. and Aquino, C. (2013). 2013 mobile future in focus. Technical report, comScore.

Looijenga, J. H. (1997). *Runes around the north sea and on the continent AD 150-700; texts & contexts*. PhD thesis, Rijksuniversiteit van Groningen.

Lunduke, B. (2012). *Illumination Software Creator: The book*. Published Electronically. Available at `http://www.lunduke.com/isc/ISC_THE_BOOK.pdf`. Accessed on 1 April 2013.

Luraghi, S. (2009). Case in cognitive grammar. In Malchukov, A. and Spencer, A., editors, *The Oxford handbook of case*, pages 136–150. Oxford University Press.

Luyten, K. and Coninx, K. (2005). UIML.net: an open UIML renderer for the .net framework. In *Computer-aided design of user interfaces IV*, pages 259–270. Springer.

Lyman, E. J. (2005). Vatican's Latin expert no stuffy academic. *USA Today*. 21 March. Available at `http://usatoday30.usatoday.com/news/world/2005-04-21-latin-foster_x.htm`. Accessed on 1 August 2013.

MacFadyen, J. (2012). Apache Cordova and Windows Phone 8. Blog post. Available at `http://phonegap.com/blog/2012/12/21/apache-cordova-and-windows-phone-8/`. Accessed on 20 April 2013.

Malchukov, A. and Spencer, A. (2009). Typology of case systems: Parameters of variation. In Malchukov, A. and Spencer, A., editors, *The Oxford handbook of case*, pages 651–667. Oxford University Press.

Mallory, J. P. and Adams, D. Q. (2006). *The Oxford introduction to Proto-Indo-European and the Proto-Indo-European world*. Oxford University Press.

Mansfield, N. (1993). *The joy of X: an overview of the X Window system*. Addison-Wesley Professional.

Marcotte, E. (2009). Fluid grids. *A List Apart*, 279.

Marcotte, E. (2010a). Responsive web design. *A List Apart*, 306.

Marcotte, E. (2010b). *Responsive web design*, chapter 3. A Book Apart.

Marsden, R. (2004). Reference grammar of Old English. In *The Cambridge Old English reader*. Cambridge University Press.

Michalowsky, P. (2008). Sumerian. In Woodard, R., editor, *The ancient languages of Mesopotamia, Egypt, and Aksum*, pages 6–45. Cambridge University Press.

Microsoft Corp. (2013a). App certification requirements for Windows Phone. Available at `http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184843%28v=vs.105%29.aspx`. Accessed on 4 January 2014.

Microsoft Corp. (2013b). MSDN developer center: Navigation patterns (Windows Store apps). Available at `http://msdn.microsoft.com/library/windows/apps/hh761500.aspx`. Accessed on 3 January 2014.

Miller, J. and Mukerji, J., editors (2003). *The MDA guide*. Object Management Group. OMG document number omg/2006/06/01. Retrieved from `http://www.omg.org/cgi-bin/doc?omg/03-06-01` on January 4th 2014.

Milne, A. A. and Lenard, A. (1960). *Winnie ille Pu*. E.P. Dutton and Co.

Ministry of Justice of Finland (1999). The constitution of Finland. Non-binding English version available at `http://www.finlex.fi/fi/laki/kaannokset/1999/en19990731.pdf`. Accessed on 20 Jun 2013.

Molnár, J. and Simon, G. (1976). *Magyar nyelvemlékek*. Tankönyvkiadó.

Motocoder (2006). Introduction of MVC structure in J2ME client. Technical report, Motorola.

Nelson, E. A. (1967). Management handbook for the estimation of computer programming costs. Technical report, Electronic Systems Division, Air Force Systems Command, USAF.

Nichols, J. (1983). On direct and oblique cases. In *Proceedings of the annual meeting of the Berkeley Linguistics Society*, volume 9, pages 170–192.

Nightingale, J. (2011). Native UI on Android. October 14. Mozilla Development Planning mailing list (dev-planning@lists.mozilla.org).

Object Management Group (2002). *Meta Object Facility (MOF) 1.4 specification*. Object Management Group. OMG document number formal/02/04/03. Available online at `http://www.omg.org/spec/MOF/1.4`. Accessed on April 28 2013.

Ogawa, A. (2009). Case in a topic-prominent language: Pragmatic and syntactic functions of cases in Japanese. In Malchukov, A. and Spencer, A., editors, *The Oxford handbook of case*. Oxford University Press.

Opus Fundatum "Latinitas" (1992). *Lexicon recentis Latinitatis*. Libraria Editoria Vaticana, Urbe Vaticana.

Oracle Corporation (2013). Java Platform SE 6: Class CardLayout. In *Java Platform SE 6 reference*. Oracle Corporation. Available at `http://docs.oracle.com/javase/6/docs/api/java/awt/CardLayout.html`. Accessed on 17 June 2013.

Oshry, M., Auburn, R., Baggia, P., Bodell, M., Burke, D., Burnett, D. C., Candell, E., Carter, J., McGlashan, S., Lee, A., Porter, B., and Rehor, K., editors (2007). *Voice Extensible Markup Language (VoiceXML) 2.1*. World Wide Web Consortium. Available at `http://www.w3.org/TR/voicexml21/`. Accessed on 18 June 2013.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Paternò, F. (2003). ConcurTaskTrees: an engineered approach to model-based design of interactive systems. In *The handbook of analysis for human-computer interaction*, pages 483–500. Lawrence Erlbaum Associates.

Paternò, F., Mancini, C., and Meniconi, S. (1997). ConcurTaskTrees: A diagrammatic notation for specifying task models. In *INTERACT 97: Proceedings of the IFIP TC13 international conference on human-computer interaction*, volume 96, pages 362–369.

Paternò, F., Santoro, C., Mantyjarvi, J., and Mori, G. (2008). Authoring pervasive multimodal user interfaces. *International Journal of Web Engineering and Technology*, 4(2):235–261.

Paternò, F., Santoro, C., and Spano, L. D. (2009). Model-based design of multidevice interactive applications based on web services. In *INTERACT 2009: Proceedings of the 12th IFIP TC 13 international conference on human-computer interaction vol. 1*, pages 892–905. Springer.

Paternò, F., Santoro, C., and Spano, L. D. (2011). Engineering the authoring of usable service front ends. *Journal of Systems and Software*, 84(10):1806–1822.

Payne, S. J. and Green, T. R. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-computer interaction*, 2(2):93–133.

Peterson, J. L. (1977). Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252.

Peyraube, A. (2004). Ancient Chinese. In Woodard, R., editor, *Encyclopedia of the world's ancient languages*, pages 988–1014. Cambridge University Press.

Phanouriou, C. (2000). *UIML: A Device-Independent User Interface Markup*. PhD thesis, Virginia Polytechnic Institute and State University.

Puder, A. (2005). An XML-based cross-language framework. In *On the move to meaningful internet systems 2005: OTM 2005 workshops*, pages 20–21. Springer.

Quirk, R. and Wrenn, C. L. (1955). *An Old English grammar*. Methuen.

Radin, M. (1918). The date of composition of Caesar's Gallic War. *Classical Philology*, 13(3):283–300.

Raju, V. and Duggi, M. R. (2008). Mobile XUL optimization. Article revision 83115. Mozilla Wiki. Available at `https://wiki.mozilla.org/Mobile/XULOptimization?title=Mobile/XULOptimization&oldid=83115`. Accessed on 1 April 2013.

Reenskaug, T. (1979). Thing-model-view-editor: An example from a planningsystem. Technical report, Xerox PARC. Available at `http://heim.ifi.uio.no/trygver/1979/mvc-1/1979-05-MVC.pdf`. Accessed on 2 May 2013.

Research in Motion, Inc. (2013). Blackberry world vetting criteria. Available at `https://appworld.blackberry.com/isvportal/downloadAWVettingCriteriaDoc.do`. Accessed on 4 January 2014.

de Rijk, R. P. G. (2008). *Standard Basque: a progressive grammar*. MIT Press, Cambridge, Mass.

Rivoal, F., Lie, H. W., Çelik, T., Glazman, D., van Kesteren, A., and CSS Working Group (2012). Media queries. Available at `http://www.w3.org/TR/css3-mediaqueries/`. Accessed on 1 May 2013.

Rix, H. (2002). Towards a reconstruction of Proto-Italic: the verbal system. In *Proceedings of the fourteenth annual UCLA Indo-European conference*, pages 1–24.

Rix, H. (2004). Etruscan. In Woodard, R., editor, *Encyclopedia of the world's ancient languages*, pages 943–966. Cambridge University Press.

Robertson, T. and Atkins, P. (2013). Essential vs. accidental properties. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Center for the Study of Language and Information, Stanford University, winter 2013 edition.

Rögnvaldsson, E. (1995). Old Icelandic: A non-configurational language. *Northwestern European language evolution*, 26:3–29.

Rossini, C. C. (1905). Noti sugli agaw. ii. appunti sulla lingua Awiya del Danghela. *Giornale della Societe Asialica Italiana*, 18:103–94.

Rounds, C. (2001). *Hungarian: an essential grammar*. Routledge.

RunRev Ltd. (2010). *LiveCode user guide*, chapter 2, pages 40–48. RunRev Ltd., 18 edition. Available at `http://downloads.runrev.com/userguide/userguide.pdf`. Accessed on 3 May 2013.

de Sahagun, B. (1585). *Historia general de las cosas de Nueva España*. Original text available at `http://archive.org/details/historiageneral02saha`. Accessed on 1 August 2013.

Sarnacki, P., Dale, T., Evans, T., Fangio, T., Pittman, J., and SproutCore 1.8 release team (2012). Getting started, part 2. In *SproutCore guides*. SproutCore. Available at `http://guides.sproutcore.com/getting_started_2.html`. Accessed on 1 August 2014.

Schmidt, D. C. (2006). Model-driven engineering. *IEEE Computer*, 39(2):25–31.

Scottish Classics Group (1996). *The Latin language*. Oliver & Boyd.

Sencha (2012). Sencha Touch 2.1.1 documentation. Available at `http://docs.sencha.com/touch/2-1/`. Accessed on 10 January 2013.

Sharp, H., Finkelstein, A., and Galal, G. (1999). Stakeholder identification in the requirements engineering process. In *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications*, pages 387–391. IEEE.

Sheppard, S. (1992). Report on the CHI'91 UIMS tool developers' workshop. *ACM SIGCHI Bulletin*, 24(1):28–31.

Smart, J., Hock, K., and Csomor, S. (2006). *Cross-Platform GUI programming with wxWidgets*. Prentice Hall.

Sofroniou, S. A. (1962). *Modern Greek*. David McKay Company.

Solomita, E., Kempf, J., and Duchamp, D. (1994). XMOVE: a pseudoserver for X window movement. *The X Resource*, 11(1):143–170.

de Souza, C. S. (2005). *The semiotic engineering of human-computer interaction*. MIT Press, Cambridge, Mass.

Spencer, A. and Otoguro, R. (2005). Limits to case - a critical survey of the notion. In Amberber, M. and de Hoop, H., editors, *Competition and variation in natural languages: The case for case*, chapter 5, pages 119–145. Elsevier.

Starosta, S. and Nomura, H. (1986). Lexicase parsing: a lexicon-driven approach to syntactic analysis. In *Proceedings of the 11th conference on computational linguistics*, pages 127–132. Association for Computational Linguistics.

Subbanna, S. and Varakhedi, S. (2009). Computational structure of the Aṣṭādhyāyī and conflict resolution techniques. In *Sanskrit computational linguistics*, pages 56–65. Springer.

Suharno, I. (1982). *A descriptive study of Javanese*. Department of Linguistics, Research School of Pacific Studies, Australian National University.

SutherlandGold Group (2011). New WixMobile helps anyone build a custom, optimized mobile site in less than 10 minutes. press release. `http://www.businesswire.com/news/home/20110803005418/en/WixMobile-Helps-Build-Custom-Optimized-Mobile-Site`. Accessed 1 April 2013.

Svantesson, J.-O. (2003). Khalkha. In Janhunen, J., editor, *The Mongolic languages*. Routledge.

Sveriges Utbildningsradio AB (2011). Giellaoahpas—liten grammatik. Available at `http://www4.ur.se/gulahalan/01\_pdf/giellaoahpas.pdf`. Accessed on 14 October 2012.

Taentzer, G. (2000). AGG: A tool environment for algebraic graph transformation. In *Applications of graph transformations with industrial relevance: Proceedings of AGTIVE'99, Kerkrade, The Netherlands*, pages 481–488. Springer.

Taentzer, G. (2004). AGG: A graph transformation environment for modeling and validation of software. In *Applications of graph transformations with industrial relevance: Proceedings of AGTIVE 2003, Charlottesville, VA, USA*, pages 446–453. Springer.

Taentzer, G. and Beyer, M. (1994). Amalgamated graph transformations and their use for specifying AGG—an algebraic graph grammar system. In *Graph transformations in Computer Science*, pages 380–394. Springer.

Thevenin, D. (2002). ARTStudio; tool for multi-target UI design. In *15th annual symposium on user interface software & technology (UIST 02)*, pages 27–30. ACM.

Thevenin, D., Coutaz, J., and Calvary, G. (2003). A reference framework for the development of plastic user interfaces. In Seffah, A. and Javahery, H., editors, *Multiple user interfaces*, pages 29–49. John Wiley & Sons.

Thimbleby, H. (2007). *Press on*. MIT Press.

Thimbleby, H., Gimblett, A., and Cauchi, A. (2011). Buffer automata: a UI architecture prioritising HCI concerns for interactive devices. In *Proceedings of the 3rd ACM SIGCHI symposium on engineering interactive computing systems*, pages 73–78. ACM.

Thomason, S. G. (2001). *Language contact*. Edinburgh University Press.

Torstendahl, S. (1997). Open telecom platform. *Ericsson Review (English Edition)*, 74(1):14–23.

Trask, R. L. (1997). *The history of Basque*. Routledge.

Trowbridge, C. R. (1907). The teaching of Latin word-order: Part I. Caesar and Cicero. *The Classical Journal*, 2(4):158–164.

Tuite, K. (2004). Early Georgian. In Woodard, R., editor, *The ancient languages of Asia Minor*, pages 145–165. Cambridge University Press.

Vanderdonckt, J., Bouillon, L., and Souchon, N. (2001). Flexible reverse engineering of web pages with VAQUISTA. In *Proceedings of the eighth working conference on reverse engineering*, pages 241–248. IEEE.

Vasu, S. C. (1891). *The Aṣṭādhyāyī of Pāṇini*. Motilal Banarsidass.

Vinokur, G. O. (1971). *The Russian language: a brief history*. Cambridge University Press.

Wade, T. (1992). *A comprehensive Russian grammar*. Blackwell.

Walker, A. T. (1918). Some facts of Latin word-order. *The Classical Journal*, 13(9):644–657.

Wallace, R. (2008). *Zikh Rasna: a manual of the Etruscan language and inscriptions*. Beech Stave Press.

Walpole, A. S., editor (1882). *Caesar: de bello Gallico I*. Macmillan.

Weiss, M. (2009). *Outline of the historical and comparative grammar of Latin*. Beech Stave Press.

Wells, J. and Draganova, C. (2007). Progressive enhancement in the real world. In *Proceedings of the eighteenth conference on hypertext and hypermedia*, pages 55–56. ACM.

Wenham, J. W. (1965). *The elements of New Testament Greek*. Cambridge University Press.

Whately, S. (1945). Noises off: Some sound-effects in Virgil. *Greece and Rome*, 14(40):17–27.

Wichmann, S. (2004). Tlapanec cases. In *Conference on Otomanguean and Oaxacan languages*, pages 19–21.

Wierzbicka, A. (1980). *The case for surface case*. Karoma Publishers.

Wierzbicka, A. (2009). Case in nsm. In Malchukov, A. and Spencer, A., editors, *The Oxford handbook of case*, pages 151–169. Oxford University Press.

Wilson, W. (1968). *An essential Latin grammar*. Macmillan.

Wireless Application Forum (2001). *WAG UAProf: Wireless Application Protocol WAP-248-UAPROF-20011020-a*. Wireless Application Forum.

Woodard, R. D. (2008). Attic Greek. In *The ancient languages of Europe*, pages 14–49. Cambridge University Press.

Wright, P. C. and Monk, A. F. (1991). A cost-effective evaluation method for use by designers. *International Journal of Man-Machine Studies*, 35(6):891–912.

Wroblewski, L. (2012). Multi-device layout patterns. Available at `http://www.lukew.com/ff/entry.asp?1514`. Accessed on 1 May 2013.

Wulf, C. (1982). Zwei Finnische Sätze aus dem 15. Jahrhundert. *Ural-Altaische Jahrbücher*, NF Bd. 2:90–98.

Yin, R. K. (2002). *Case Study Research: Design and Methods*. Sage Publications, third edition edition.

Zuazo, K. (1995). The Basque Country and the Basque Language: An overview of the external history of the Basque languages. In Hualde, J. I., Lakarra, J. A., and Trask, R. L., editors, *Towards a history of the Basque language*. John Benjamins.

Appendix A

Classification of industrial tools

Mobile Vision's report divided up the 53 cross-platform tools they surveyed into five categories, which were summarised in chapter 2. The tools surveyed in these categories were:

JavaScript toolkits: DHTMLX Touch; impact.js; iUI; JoApp; jQuery Mobile; NetBiscuits; Sencha Touch; SproutCore; The M Project; Wink Framework. Adobe Flex is related to this category, being an ActionScript/AIR toolkit.

App factories: Magmito; iBuildApp; MobileNationHQ; Mobjectify; Verivo Platform; Red Foundry; DragonRAD; WIX Mobile.

Web-to-native wrappers: apparat.io; KonyOne; Exadel Tiggzi; FeedHenry; Application Craft; WOPE; PhoneGap.

Runtimes: AMP Studio; Antix Games Studio; Adobe AIR; Corona; Titanium; EDGELIB; QT; RhoElements/Rhodes; LiveCode; SIO2 Engine; Mobinex Smartface; Spot Specific; TotalCross; Unity Engine; Unreal Engine; Moai; Aqua Platform.

Code translators: webMethods Mobile Designer; Marmalade; Illuminations Software Creator; MonoTouch/Mono for Android; XMLVM; iFactr Monocross; J2ME Polish; MoSync.

When considering software architecture, only the part of the software which resides on the mobile device itself is considered, rather than any larger system of which it is a part. In addition, only software architectures which are explicitly named in the software and documentation are counted. Some of the tools act as libraries of physical components and other tools, claiming to provide these in an architecture-neutral way. These tools are counted as "Neutral". Others are purely designers and to these the structure of the code is immaterial, or they are "codeless". These, along with XMLVM, which is purely a bytecode translator, are

counted as "n/a". Still others are focussed on games, and provide architecture support based on agents, controllers, and objects in a game world. These are counted as "Game architectures".

MVC and variants: Sencha Touch; SproutCore; The M Project; RhoElements/Rhodes; MonoTouch; Titanium (via the inbuilt Alloy library). QT versions since 4.0 use a simplified Model/View architecture.

Other defined architectures: Mono for Android uses the Android "Activity" model. JoApp uses an architecture based on the Observer pattern. Illuminations Software Creator uses a visual programming/flowchart based architecture.

Game architectures: impact.js; EDGELIB; SIO2 Engine; Unity Engine; Unreal Engine; Moai; Marmalade.

Architecture-neutral: Adobe Air; DragonRAD; DHTMLX Touch; iUI; jQuery Mobile; NetBiscuits; Wink Framework; Corona; LiveCode; SmartFace; TotalCross; AppMobi; Worklight; J2ME Polish; MoSync; Exadel Tiggzi; FeedHenry; Application Craft; WOPE; PhoneGap.

N/A: Magmito; iBuildApp; MobileNationHQ; Mobjectify; Verivo; Red Foundry; Wix Mobile; Spot Specific; XMLVM.

Classifying the tools according to the Arch model (extended with a "platform" component as defined above) was done according to the following criteria:

Platform: These tools must provide either an abstraction layer between the hardware and the software such that an app can be deployed onto multiple platforms, a deployment mechanism to get the application onto those multiple platforms, or both. Tools which fall into this category are: QT; The M Project; RhoElements/Rhodes; MonoTouch/Mono for Android; Appcelerator Titanium; iBuildApp; Mobjectify; Verivo; Red Foundry; Spot Specific; XMLVM; Adobe AIR; Corona; LiveCode; SmartFace; TotalCross; AppMobi; MoSync; FeedHenry; WOPE; PhoneGap; impact;js; EDGELIB; SIO2 Engine; Unity Engine; Unreal Engine; Moai; Marmalade; Illuminations Software Creator.

Physical presentation: These tools must adapt the physical components to fit in with the native appearance and behaviour of those components, either by using the native controls or by a process of styling non-native components. Tools which fall into this category are: QT; The M Project; RhoElements/Rhodes; Titanium; iBuildApp; Verivo; Red Foundry; Spot Specific; XMLVM (by means of its reimplementation of the Android UI library); DragonRAD;

NetBiscuits; LiveCode; Smartface; TotalCross; MoSync; Marmalade; Illuminations Software Creator; Touch; SproutCore; DHTMLX Touch; iUI; jQuery Mobile; Wink Framework; J2ME Polish; JoApp. Corona falls into this category in a limited way, providing native text controls only.

Logical presentation: These tools must allow components to be replaced with other components based on the platform or form factor in use. Tools which fall into this category are: QT (by means of its semantic QML elements); NetBiscuits; Sencha Touch (by means of using its Profiles feature).

There are no tools in the survey which provide adaptation of any Arch layer above logical presentation.

Appendix B

Further data on languages surveyed

B.1 Distribution by language family

To demonstrate that case is a real phenomenon in language and not just a chance resemblance, it is necessary to demonstrate that the the languages that exhibit case did not all get it from one place. Besides innovation within a language, t here are two main ways in which languages can pick up constructs: by inheritance from a parent language, or by borrowing from another language with which they are in contact. This section and the next deals with each of those ways in turn.

Languages change and divide over time. Because of this, they can be arranged into "family trees" which trace their ancestry. French, Spanish and Italian, for example, all are descendents of Latin. A "language family" consists of all the languages which are descendents of a common ancestor language. The "Romance" language family, for example, contains all the languages which share Latin as an ancestor (figure B.1; note that these diagrams are an extreme simplification of reality, since they do not show intermediate stages in linguistic evolution between the ancestor and the descendent, or the possibly non-homogenous state of the ancestor language).

The common ancestor of a language family is not always known or attested in writing. If this is the case, then the theoretical last common ancestor is called a "proto-language", and is named after the language family. In Italy at the time of the Romans there were a number of related languages, such as Umbrian, Faliscan and Oscan (Fortson, 2009, chapter 13). These, along with Latin and the descen-

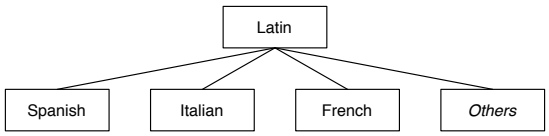


Figure B.1: The Romance language family

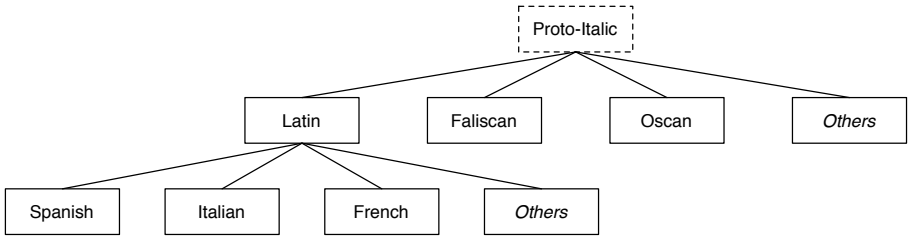


Figure B.2: The Italic language family

dents of Latin form a language family called the Italic languages. The last common ancestor of all these languages existed before writing arrived in Italy, and so it is unattested. Therefore, linguists construct a theoretical last common ancestor, called "Proto-Italic" (Rix, 2002). The Italic language family, including its proto-language, is illustrated in figure B.2.

Because of their common ancestry, languages which share a language family tend to have grammars which resemble each other. To demonstrate that case is not a feature of a single language family, the language families that contain the languages being discussed are listed below.

The Afro-Asiatic languages form a large language family containing around 300 living languages, largely spoken in north Africa and the Middle East (Lewis et al., 2013). Representatives of two of the major subfamilies of Afro-Asiatic are represented in the languages used in the survey: Arabic is a member of the Semitic family, and Awngi is a member of the Cushitic family.

The Indo-European languages form another large language family, containing about 400 living languages (Lewis et al., 2013). It includes nearly all of the languages current in Europe and many of those in use in India and in Iran. It is the language family which includes Modern English (Fortson, 2009, and Mallory and Adams, 2006 give overviews of the family). Six subfamilies of Indo-European are represented in the languages used in the survey: Old English and Icelandic are members of the Germanic subfamily; Greek is essentially a subfamily on its own; Latin is a member of the Italic subfamily; Russian is a member of the Slavic subfamily; and the Tocharian languages form a subfamily of their own.

The Kartvelian languages form a small language family spoken in the Caucasus, mostly in Georgia (Gamkrelidze, 1966). In the survey, Georgian represents this language family.

The Mongolic languages form a family of closely-related languages spoken in and around Mongolia (Janhunen, 2003a). Khalkha Mongolian, the official language of Mongolia, represents this language family in the survey.

The Nakh-Daghestanian languages (or the *North-east Caucasian languages*) form another language family used in the Caucasus. There is likely no genetic connection between the Nakh-Daghestanian languages and the Kartvelian languages, despite their geographical proximity. One sub-group of the Nakh-Daghestanian languages is known for its rich case systems, being briefly listed in the Guinness book of Records as having the richest case system in the world (Comrie and Polinsky, 1998). One of these languages, Tsez, represents the family in the survey.

The Oto-Manguean languages form a language family spoken widely in Mexico (Lewis et al., 2013). This family is represented in the survey by Azoyù Tlapanec.

The Pama-Nyungan languages form a large language family which cover most of the continent of Australia. This family is represented in the survey by Kalkatungu, although other Pama-Nyungan languages have similar case systems (Blake, 1977).

The Turkic languages form a language family which spreads over a vast area from the Mediterranean in the west to Siberia in the east (Lewis et al., 2013). The Turkic languages are represented in the survey by Turkish.

The Tyrsenian languages are an extinct language family consisting of three closely-related languages which were spoken in Italy, in the Alps and on the island of Lemnos (Rix, 2004). Etruscan is the most well-attested; the other two languages (Rhaetic and Lemnian, respectively) are extremely fragmentary.

The Uralic languages form a language family widely spoken in Europe (Austerliz, 1989). The Uralic languages are represented in the survey by Hungarian and Finnish.

Languages isolate are those which cannot reliably be connected to any other language. Two languages isolate are represented in the survey: Basque and Sumerian.

Two large language families are not represented. The first of these is Sino-Tibetan, the large language family that includes the Chinese languages and the Tibetan ones. Case is very uncommon in this family. A notable exception can be found in Nepal Bhasa, also known as Newari, which has a rich case system (Hale and Manandhar, 1973). It is not included here for two reasons. First, there is a vast shortage of reference grammars of the language that are not written in

Nepal Bhasa itself (Hargreaves, 2006). Secondly, it has been strongly influenced by Indo-European languages—specifically, those spoken in India and Iran—and thus it cannot really be taken as a representative example of the language family.

The other large language family missing is the Niger-Congo family, which is another large language family that is not prone to exhibiting case. Its geographical distribution is restricted to sub-Saharan Africa, and König (2008) notes that case is rare in Africa.

B.1.1 Borrowing

In his introduction to *The World's Major Languages*, Bernard Comrie (1987) makes the point that if two languages come into contact, any number of elements may be borrowed by one from the other. English speakers are likely to be aware of this already, as English is riddled with borrowed vocabulary. Learned terms borrowed from Latin, Greek and Arabic are perhaps the most obvious, such as "geometry" from Greek, "algebra" from Arabic and "femur" from Latin. Other borrowings have been more completely absorbed into English: the word "beef" comes from Norman French (Barnhart, 1999, p. 85); and the name of the drink "punch" may come from an Indian word meaning "five", referring to the number of ingredients (Barnhart, 1999, p. 863).

These examples are all vocabulary. Larger systems can be borrowed too. For example, English has borrowed a number of productive prefixes from Greek and integrated them into the language, as can be seen in the word "television". This word consists of a prefix borrowed from the Greek (τῆλε-, "afar" or "at a distance"; see Liddell et al., 1940) and a Latin root *vis-* meaning, roughly, "seeing". What has been borrowed here is not a whole word; the word "television" certainly never appeared in either of its parent languages. Instead, what has been borrowed is a prefix and a rule that says that this prefix can be combined with other words.

Even larger patterns of borrowing are possible: English borrowed its third person plural pronoun system from Old Norse (Barnhart, 1999, pp. 1131 and 1133), and Japanese borrowed much of its numeric system from Chinese (Ifrah, 2000).

In general, the larger the system, the less it is likely to have been borrowed. To find that a language had borrowed its entire case system from another would be startling, but by no means impossible. Case systems can certainly be partially affected by borrowing: for example in some outlying dialects of Greek, the case system is being re-formed along more Turkish lines through contact with Turkish (Thomason, 2001, ch. 4). However, if the languages concerned are numerous enough and spread widely enough in time and space, the probability of any similarities being the result of borrowing becomes near zero.

Figure B.3 shows the temporal distribution of the languages discussed over history. The temporal and geographical distribution of just these twenty languages makes widespread borrowing unlikely. When the enormous number of other languages that exhibit case are taken into account, the probability that all case systems are borrowings from one another becomes vanishingly small.

B.2 The ages of languages

The age of a language is a difficult thing to measure. The two major difficulties in this area raised by the set of languages surveyed are:

- Languages change gradually and slowly. There is no one point, for example, that one can take as the end of Latin as a vernacular tongue. Instead, over a period of several hundred years, Latin slowly became the Romance languages. This process of slow change is especially true of Icelandic, which emerged slowly from Old Norse, and Russian, which emerged slowly from Old East Slavic.

- Many languages in the world still have no writing system; and many languages, both ancient and modern, gained their writing systems some time after the language itself was commonly spoken. Spoken language leaves few traces: it is extremely hard to tell how long a language existed before its first written evidence.

To deal with these problems, the dating given uses a number of rules. The starting date is taken as the earliest of:

- The date of the earliest written evidence in the language, or a recognizably modern form of the language if there is writing all the way through a period of strong language change.

- The earliest date that the language is discussed by linguists, anthropologists or explorers.

- If there is, or was, a separate but related written or elite language or dialect which was discussed or attested earlier than the language discussed, then that written dialect is used instead in the above criteria.

If the language is extinct, the ending date is taken as either:

- The last known evidence that shows the language in use *as a living language*. The caveat is important—both Sumerian and Latin were in use as liturgical, formal languages long after their extinction as spoken languages.

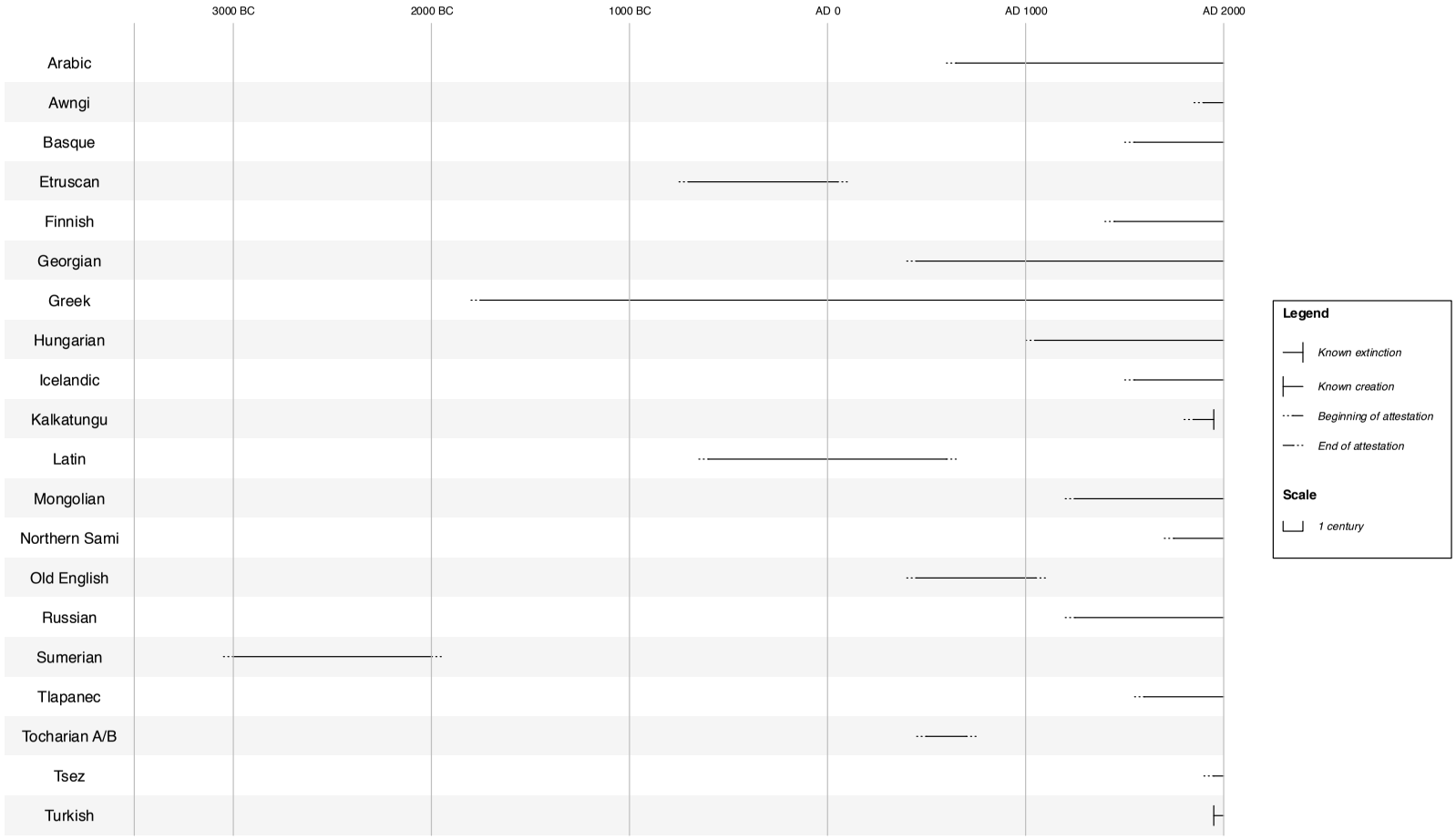


Figure B.3: Distribution of languages discussed in time.

- The death of the last known speaker of the language.

These rules are unfair to languages that have no writing system and which have been documented only recently, making them look far younger than they may well be.

Arabic is treated here in its Classical Arabic dialect. There is some argument as to whether Classical Arabic and Modern Standard Arabic are the same or different languages; Arabic sources tend to treat them as two registers of the same language, whereas English-language sources tend to treat them as separate. Here, to give a fairer view of the age of the language, and because of the similarity in their case systems, the two are treated as a single language. Classical Arabic is attested from around 700 AD (Holes, 2004). Modern Standard Arabic is still in use.

Awngi was first documented by Rossini (1905). It is still in use.

Basque "opened its doors to history in the 16th century" (Zuazo, 1995). Trask (1997) notes that the grammar in texts from this period is nearly the same as in modern Basque. Basque is still spoken is still in use.

Etruscan is known only from inscriptions and from a few longer texts. The earliest inscription is c. 700 BC; the latest around 50 BC (Bonfante and Bonfante, 2002; Rix, 2004; Wallace, 2008).

Finnish is attested first in a manuscript from around 1470 (Wulf, 1982). It is still in use.

Georgian became a written language around the middle of the fourth century AD (Tuite, 2004). It is still in use.

Greek has changed surprisingly little over time. The earlier date given here, of around 1800BC, is from Fortson (2009). Modern Greek is still spoken.

Hungarian is attested earliest just after 1000 AD (Molnár and Simon, 1976).

Icelandic is continuously attested throughout its transition from Old Norse, which makes it difficult to give a firm date. The date given here—1540—is the date of publication of the first translation of the New Testament into vernacular Icelandic (Rögnvaldsson, 1995). Icelandic is still in use.

Kalkatungu is, according to Blake (1979), first discussed by settlers in Australia in 1886. Lardie Moonlight, the last fluent speaker of Kalkatungu, died between 1979 and 1983 (Blake, 1983).

Latin is here shown as being a living language from around 650 BC to around 650 AD. These dates are following Weiss (2009).

Mongolian is here represented by the Written Mongol dialect or language rather than the Khalkha dialect or language. It is controversial, in the Mongolic language family, where dialects begin and languages end. Written Mongol is the old formal, written language, which is still in use today by the Mongolian-speaking peoples of China. This is first attested around 1250 (Janhunen, 2003b).

Northern Sami as distinct from Sami as a whole was first discussed in 1748 by the Norwegian linguist Knud Leem (Bergsland, 1950; Garcia-Ontiveros, 2011). It is still in use.

Old English is first attested in short runic inscriptions. Looijenga (1997) dates the earliest of these to around 450 AD. Old English becomes Middle English, and loses its case system, under the influence of Norman French, starting around the middle of the 11th century AD (Quirk and Wrenn, 1955).

Russian The Russian language became distinct from its parent language in the 14th century (Vinokur, 1971).

Sumerian is attested as a living language from around 3000 BC to around 2000 BC, although it was used as liturgical language afterwards (Jagersma, 2010).

Tlapanec seems to have been something of a mystery until recently. It is first mentioned as a language in the *Historia general de las cosas de Nueva España* of Bernardino de Sahagun (1585), but is not described. As late as the late 19th century, Hubert Howe Bancroft could still say "Wedged in between the Miztec and Zapotec are several tongues, of which, excepting a few Lord's Prayers, I find nothing mentioned but the names ... there are mentioned the Chatino, Tlapanec, and Popoluca" (Bancroft, 1882, p. 752). The date of the earlier mention is used here, although neither of these refer to the specific Azoyú dialect or language (again, here, the distinction between language and dialect is controversial). Azoyú Tlapanec is still in use.

Tocharian A and B remain to linguists in the form of religious texts written between 500-800 AD (Fortson, 2009). Very little is known about the speakers of the language.

Tsez was first discussed in the middle of the 20th century (Job, 2004).

Turkish as spoken today is a constructed language based on the earlier Ottoman Turkish; but it sufficiently different to be considered a different language. It appeared during the rule of Ataturk, in 1932. It is still spoken today.

Appendix C

Applications built by other developers

This appendix presents the details of the applications and comments from developers in the self-directed development study, presented in Chapter 7.

C.1 Participant 1

In addition to his professional practice as a developer, participant 1 ran a small publishing company. In his capacity as a publisher, he visited trade shows and other such industry events, and on those occasions found it limiting to have to get his laptop out in order to get information about the books that the company had in production, such as dates when things would be published or the expected prices of books. The company already kept this information in a database that was accessible through a web service, so the participant felt that it would be a good trial for the AppMaps tool and case system.

The purpose of the application was to let the participant answer several questions quickly:

- Given a book, how far through the publication process is it? What details have already been decided about it?
- Given a stage in the publishing process, how many books are in that stage?
- Which books are waiting on author input, and which on publisher input?

The application map is shown in figure C.1. In the initial state, the user has a list of all the books currently in the publishing process (that is to say, all the books in the publisher's catalogue that aren't out of print). This screen also lets the user filter by books that are published, those that are in production, and those for which essential data is missing from the database, as well as filter based on specific stages in the publication process (see figure C.2; note that all the screenshots of this application are using placeholder data, as the data that the application is actually used with is confidential).

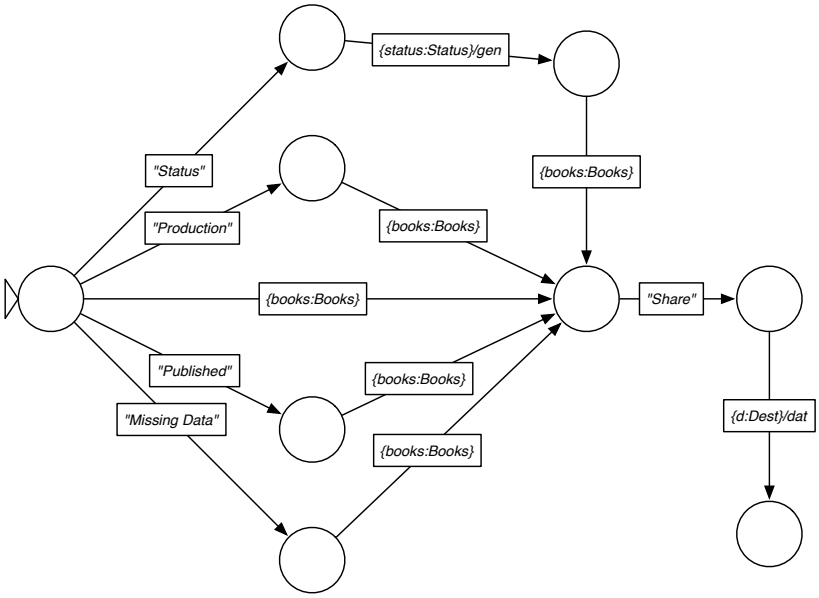


Figure C.1: Publishing company application map

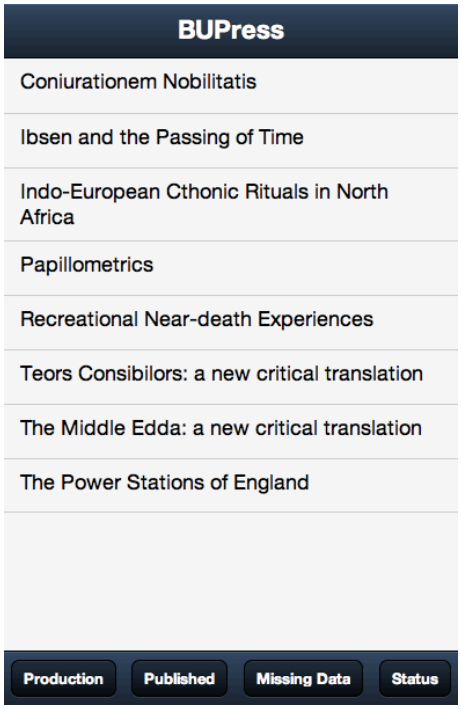


Figure C.2: Publishing company front page

Figure C.3: Publishing company list of categories

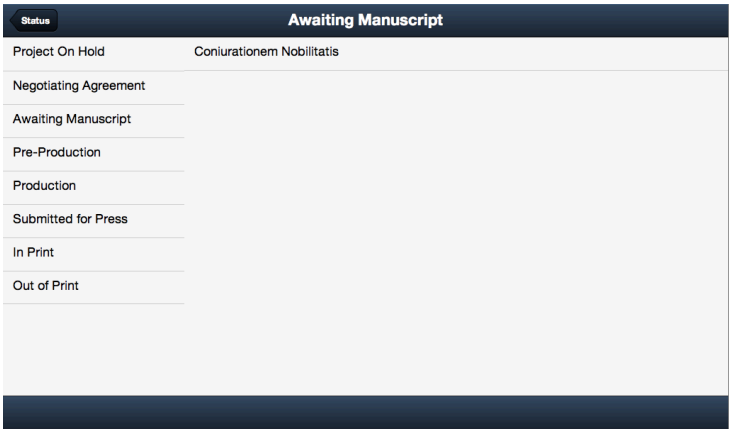


Figure C.4: Publishing company list of statuses and books on tablet

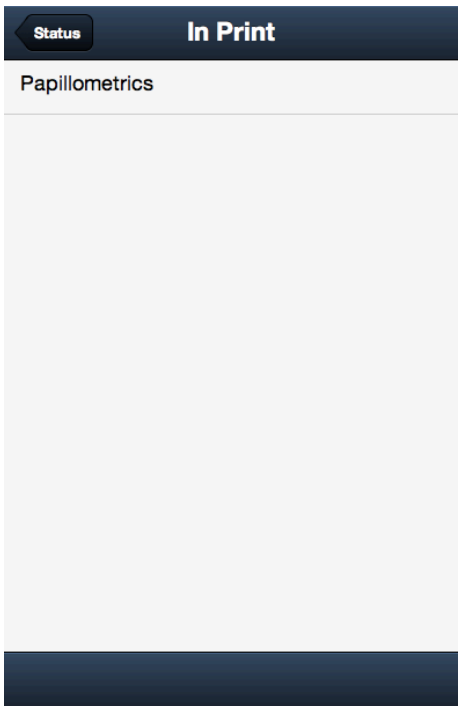


Figure C.5: Publishing company list of books

Most of these buttons just lead to another list of books, but the button that filters by the book's status leads to a list of possible statuses for the book (figure C.3), and then each of these gives rise to a list of books (figure C.5 and figure C.4). The list of statuses is a genitive edge: the participant said that it fit his mental model of the process that these were categories of books. The resulting interface is compliant with the human interface guidelines on both iOS and Android phones and tablets.

The viewing of a book gives the user the choice to share it, and when this option is chosen, a destination must be chosen. This destination is marked as dative. Note that the destinations here are not social networks but are instead other people within the company.

The participant designed the application map in two phases: in the first, he designed the application map for an ideal phone. He then built the views, and checked that they worked well in the ideal phone application map. After that, he went through and added genitive and dative markers to the appropriate edges in the application map, after which he checked that the views worked in that context, found that they did, and declared the application complete.

None of the views perform any plasticity of their own, nor in this case were any device-specific modifications made to the stylesheet. All the plasticity that the

participant felt necessary for the application was managed by the default stylesheets for the platform and by the cases on the edges of the application map.

Participant 1 did not feel that his application was, in his words, "production-ready". He had reservations about the quality of the physical layer of the interface as a problem, especially with regards the typography: he also noted that his application consisted almost entirely of list boxes, which he considered possibly not the best user interface control for the application. He had chosen these because he "didn't have the time [during the study] to properly investigate other alternatives". He did, however, say that the dialogue component met his quality standards and that the application as a whole was a "very useful prototype".

He mentioned that case had provided a marked efficiency gain compared to his normal practice while "making the transition between concept and prototype very rapid". This early phase of the development process was the one at which he had found case most useful; as an application would progress towards production, he would have expected to spend more and more time "concentrating on what it looks like on different platforms from a design perspective". The level of plasticity provided was what was "needed" for the application because "it added the ability to adapt between [form-factors] without making me do much extra thinking".

Participant 1 felt that the case system was "very definitely" worth learning in terms of the efficiency gains he had experienced during the development process. He thought that case as it was implemented here was applicable to all the mobile platforms he had used. He thought that case would work well in the kind of application that he built (by which he meant data-driven) but did point out that in the AppMaps tool there was no "built-in way to override the case system", and that if this were added then the case system would be potentially applicable to any data-driven application he could think of. He also was of the opinion that there was no reason why case conflicted with the industry domain in which he was developing applications, nor any of the domains for which he had built applications or mobile websites in the past.

C.2 Participant 2

The application that participant 2 built was also, by chance, book-themed. Participant 2 was a keen book collector, who had previously catalogued all their books in LibraryThing, which is an online book catalogue. He wanted to make an application for himself and some likeminded friends which could be used while in a bookshop or otherwise away from home.

The application map for the application is given in figure C.6. The user's initial entry to the application is a search form. The user can either fill this form in and initiate a search, or can choose to display books that they have not yet read,

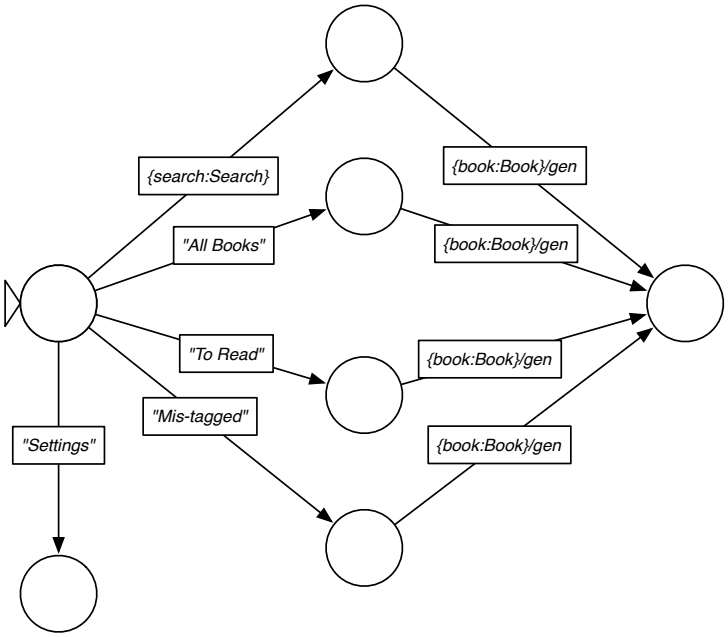


Figure C.6: Book catalogue application map

books that are wrongly tagged (for this application, this means that the catalogue does not specify whether they are physical books or electronic books), or just to display all books (figure C.7). Choosing any of these options takes the user to a list of books: the edges that correspond to choosing the books are all marked as genitives. At this point the user can view the details of the book (figure C.8). From the front page, the user can also enter a settings view: this is so that they can put in their LibraryThing username and API key (figure C.9).

There is no use of the dative whatsoever in this application: participant 2 stated that he understood what it did, he had just found no use for it in the application that he was building.

Participant 2 used case as he went along to plasticise the application, rather than going back and adding case in a separate pass. As each edge went in, he considered what case it should be in, then created views to implement the application map, then went back and checked and reconsidered the case assignments to edges. The major reconsidering that he undertook involved thinking hard about whether to make the "search" a genitive edge, given that it produced a subset of books. Eventually, he made the decision to leave it as a non-genitive edge, as reflected in the final application map. This was because he definitely wanted to have the choice of book as a genitive edge, and because there was an issue with the software which prevented multiple genitives working correctly (the same issue that arose in the Agritechnik application; see section 6.6.5).

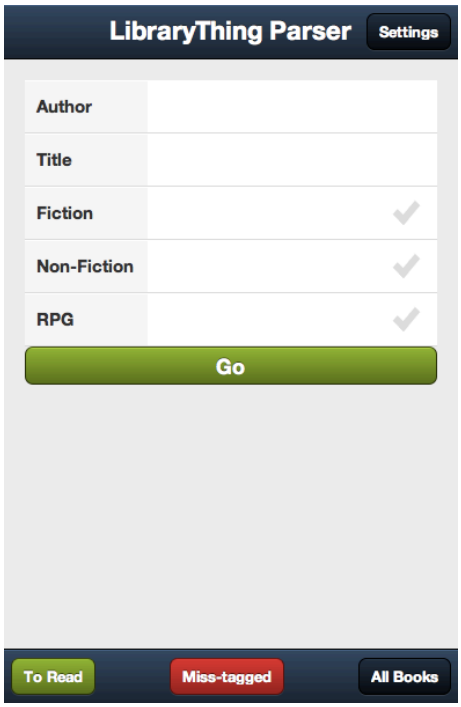


Figure C.7: Book catalogue: search screen

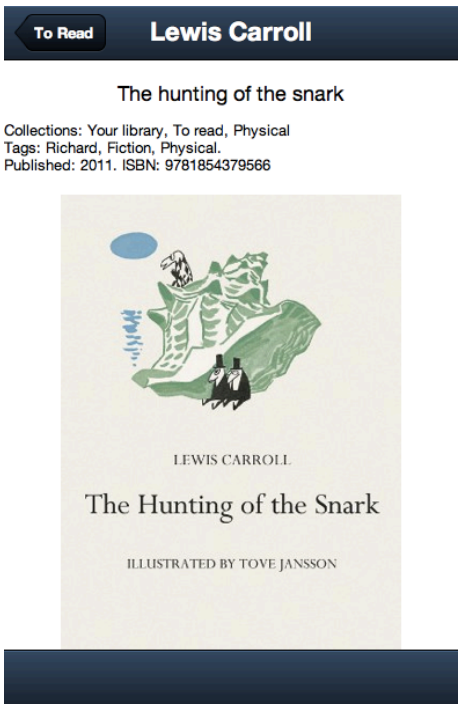


Figure C.8: Book catalogue: book details

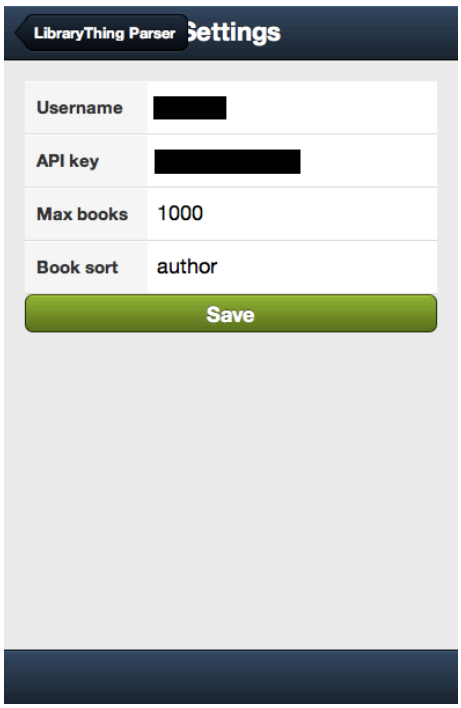


Figure C.9: Book catalogue: settings screen

None of the views perform any plasticity of their own. A device-specific stylesheet was used to modify the physical appearance of the search form slightly depending on whether it was running on a phone or a tablet. All other plasticity was managed by the default stylesheets and by the cases on the edges of the application map.

Participant 2 stated that he was "very happy" with the quality of the resulting user interface was "sufficient for my needs especially given the time spent making it". He said that it sped up the development process in general compared to his previous cross-platform mobile efforts, but that the big time-saver was predictability: when he added an edge to the application map with a case on it, he felt that he knew that the system was going to do what he expected it to: it "made it easy to know what behaviour to expect from the system for a given type of link/state transition". He said that it had been worth learning the case system for the efficiency gains, but felt that he had intuitively understood the concepts anyway, so not a great deal of learning was called for. He felt that the case system was well-suited for development on the current set of mobile platforms, said that he could think of "no reason" why it wouldn't be well-suited to use in any other application whose purpose was primarily to do with browsing and editing data as his was, although he noted that the demands his application put on the user interface tool were scarcely intensive. He felt that what industry an application was aimed

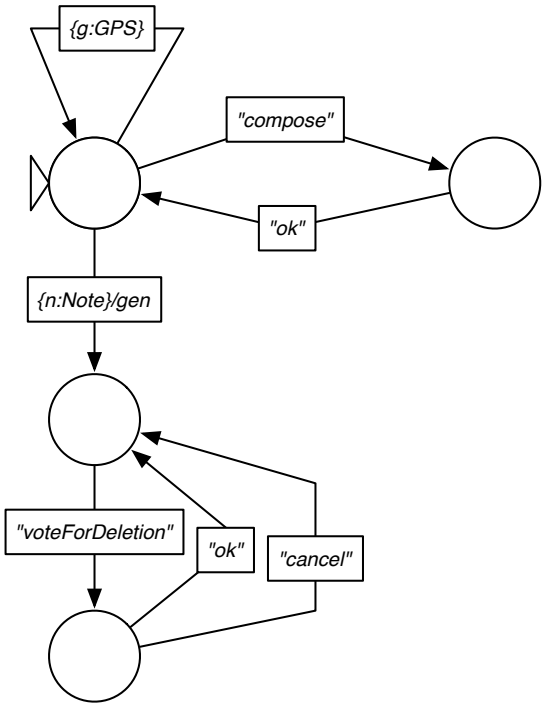


Figure C.10: Geotagged social media application map

at was irrelevant to the success or failure of case as a way of developing an application, and felt that whether it was data-driven or not was a much more important criterion.

C.3 Participant 3

Participant 3 designed a social media application which allowed people to leave public textual notes at physical places. Whenever the user opened the application they would receive a list of the notes that had been left within a certain distance of them, along with the directions they were in. Moderation was done by people voting for the deletion of a note.

The application map for the application is given in figure C.10. In the start state the user can select a note: this note is in the genitive, as something whose details will later be displayed (figure C.11). The user may also create a new note. The GPS edge on the initial state is a pseudo-view, which permits user input while having no visual component. In this case, the user input in question is the user's movements over the face of the earth, as measured by the phone's satellite navigation system. As the user moves around, the list of notes that they can choose from changes.

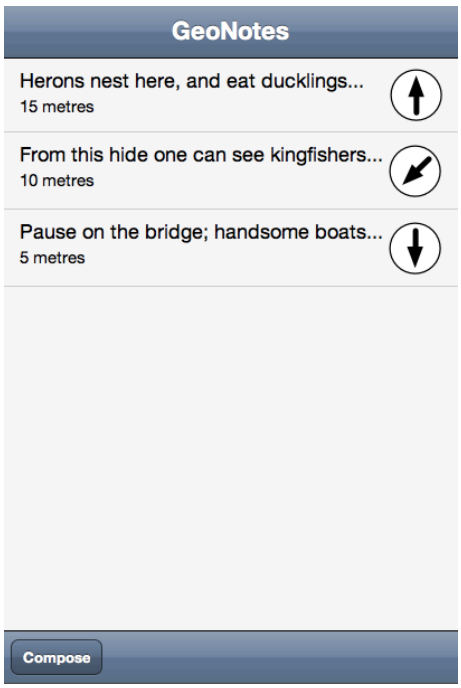


Figure C.11: GeoNotes: notes list

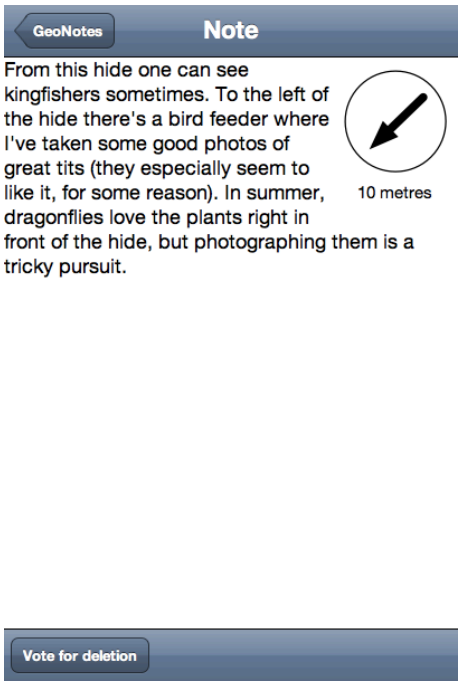


Figure C.12: GeoNotes: viewing a note

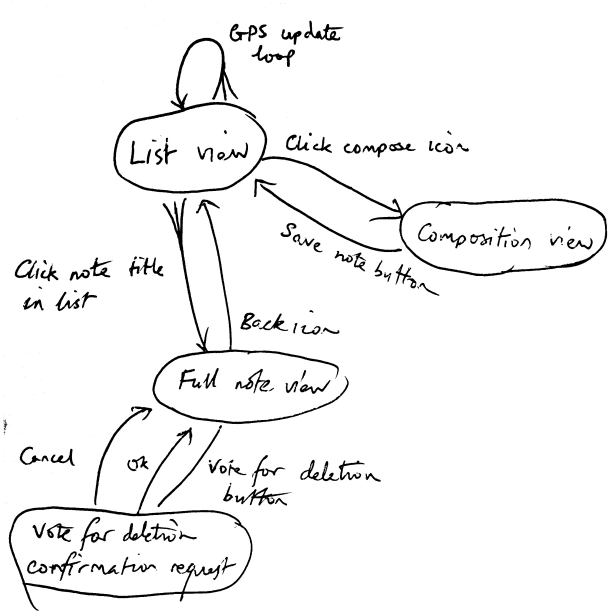


Figure C.13: Geotagged social media hand-drawn application map

When the user has chosen a note they can read it, and they may have the option of voting to delete it (figure C.12). When they choose to vote for deletion they need to confirm their vote.

As is shown by his original draft of the application map on paper (figure C.13), participant 3 first designed the application map without cases, and added them on a subsequent pass. With regards to this, he said that for him the design of the application map for the ideal phone and its subsequent plasticisation were two separate tasks during the development process.

None of the views provided any plasticity of their own: nor was anything but the standard stylesheet used. All plasticity was managed by the default stylesheets and the case on the edge.

Participant 3 was not particularly pleased with the quality of the interface he created, mainly criticising the physical aspects of the interface. He stated that the quality of the interfaces of HTML-based applications suffers because the developer is "beholden to how a browser renders the toolkit I'm using" and because it does not provide access to "more powerful visual features such as 2D [and 3D] graphics libraries", and that he tends to have a "a predisposition to go for a native framework such as Cocoa Touch" but acknowledged that this was partly due to prejudice on his part. The dialogue structure and the parts of the physical interface mediated by case were of a quality he was happy with, except for the unpredictability and low rendering quality forced on him by the browser component.

Case had saved time for him by allowing him to explore alternatives more quickly: "once the data structures were defined, I could try out different views over them and see how these would produce use cases." He added later that "I have been on many GUI app development processes where one person's idea of what the metaphor to represent the data should be makes it harder for the users to understand what they're looking at, because it is greatly at odds with their own... I think [the AppMaps case system] succeeds in doing this because it makes the [exploratory] process of building a picture of a concept explicit." Several days later, he added: "I think I've done it a bit wrong because I can see how appmaps is particularly useful if you're actually using it in the early stages, given a data structure, to play with ways to define a view on it. Whereas I thought [how the application should work] up in advance."

He felt that the case system would definitely be worth learning for the efficiency gains he had experienced: "it lets you explore how a data structure produces a use case, so if that's the way you are approaching your app design (which I think is common - lots of applications, especially mobile ones, are built to provide a view on some currently existing data source) it makes it easy to figure out how to do it in a usable way."

He was unable to think of any mobile platforms on which the concept of case was not applicable and added that he saw no reason why it should be restricted to mobile platforms at all. He though that the concept of case fitted well with the kind of application he had built, and that the kind of application was more important than the industry that it was in.

C.4 Participant 4

Participant 4's application was a to-do list application. The application stored a list of tasks locally on the mobile device in a database, and allowed the user to edit tasks, add notes to those tasks and mark them as complete. Participant 4 did not finish building her application.

The application map for the application is given in figure C.14. The user begins in a state where they can either choose an existing task, choose to filter the task list, or choose to add another task. Once they're in a task, they can either mark it as complete or delete it.

Participant 4 used the genitive in both for categories and for its partitive (see section 5.3.3) meaning. She said that tasks, when they appeared in the application map, were in the genitive because the details of the task was what the application was focussing on: the categories and priorities are subsets of the lists of tasks. This matched the model exactly but actually produced a "very strange" concrete user interface on the tablet due to a software issue (the same software issue as

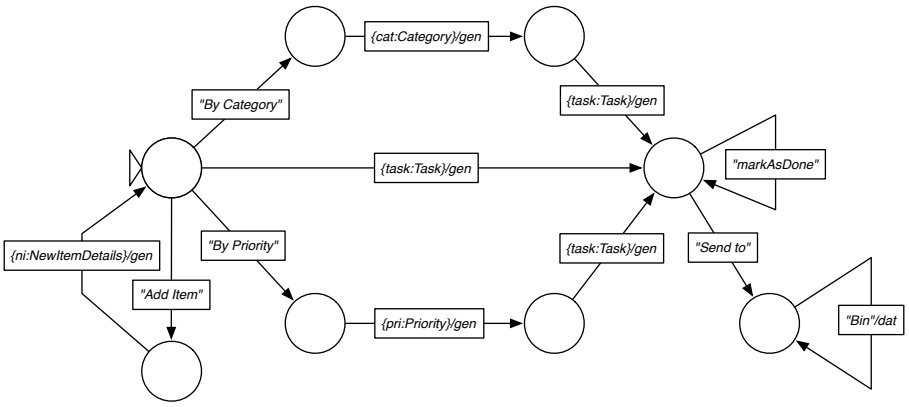


Figure C.14: To-do list application map

described in section C.2 and section 6.6.5). She (correctly) identified this as a bug in the software because it did not behave as her understanding of the genitive case did; and rather than making her software incorrect according to her understanding of the model informed the author firmly that she would wait for a newer version of the software.

Datives were used for logical destinations within the software, rather than for destinations outside of the software. The only destination that made it into the final version of the software was a "rubbish bin" for deleting items.

Participant 4 used the cases to label edges as she went along. Rather than creating the application map as a unified whole and then filling it in with views, however, she started with a small, seed application map consisting of the data viewing part of the application, and created all the views for that; after this, she added on extra features, adding each to the application map first along with its attendant case labels and then to the code. She felt that she understood what the cases meant, but not necessarily how to apply them: this was due, she thought, to ambiguities in the documentation for the tool that she had been given.

Participant 4 was not happy with the quality of the user interface that she produced, because her application was unfinished and did not work. She had hit far more software bugs in the AppMaps tool than any other participant or the author, and felt that this also contributed to her not finishing her application. She also had doubts about the quality of the physical interface that the underlying Sencha Touch toolkit had produced, stating that it didn't "look good or feel good. Buttons looked wrong and scrolling was jerky".

She did, however, feel that case was "very helpful for segregating modes of interaction with the application and making them easier to style appropriately across form factors". By "modes" here she did not mean modes as the word is used in chapter 2: instead, she said that she meant that each case represented a

coherent way of interacting with the machine independent of what data it was that the application was actually doing. She also highlighted that case would "probably also be quite useful as a means of encouraging the developer to see the 'big picture' of user interaction with the application early in its development".

She did feel that the case system would be worth using for applications that needed to be present on multiple form factors, and felt that possibly it did not go far enough: she said she would be "curious to know whether the cases can be extended beyond genitive and dative however, as these only cover a small range of interactions." She especially noted the absence of a "tool" or "instrument" case.

Participant 4 did not identify any areas in major mobile platforms where the approach would not be applicable. She thought that her application area contained a number of "modes" (in the sense used above in this section) that were not covered by case.

C.5 Participant 5

Participant 5's application was a prototype of an application to let users browse a crowdsourced data set about access for disabled people at venues local to them. The application used the GPS position of the user to show nearby points of interest on a map, and then allowed them to look at the accessibility of those points of interest.

The application map of the application is presented in figure C.15. At the start, the user is presented with a map centred around their current location (figure C.16). They can either view details of points of interest around them, use the "go to" button to move the map to another location or search for points that match certain criteria (figure C.17). These points are then displayed on another map view (figure C.18).

Participant 5 said that he used the genitive in a categorial sense. It is the edge that corresponds to the search criteria that is in the genitive, and his reasoning for putting this edge in the genitive was that by entering search criteria the user would be effectively creating a categorisation of the data points to meet their current needs. He used the dative to let users share the details of points of interest on the map; this dative used the built-in sharing controls to share to social media and to email.

Participant 5 was happy with the quality of the interface that he had produced, saying that for the things that he was building the physical and cosmetic aspects of the interface were usually simple. He said that he could easily have turned the prototype into a "real-world app", and that the reason that the app had remained as a prototype was only that of time constraint. He stated that "the reduction in implementation required [using the case system] to develop for multiple platforms

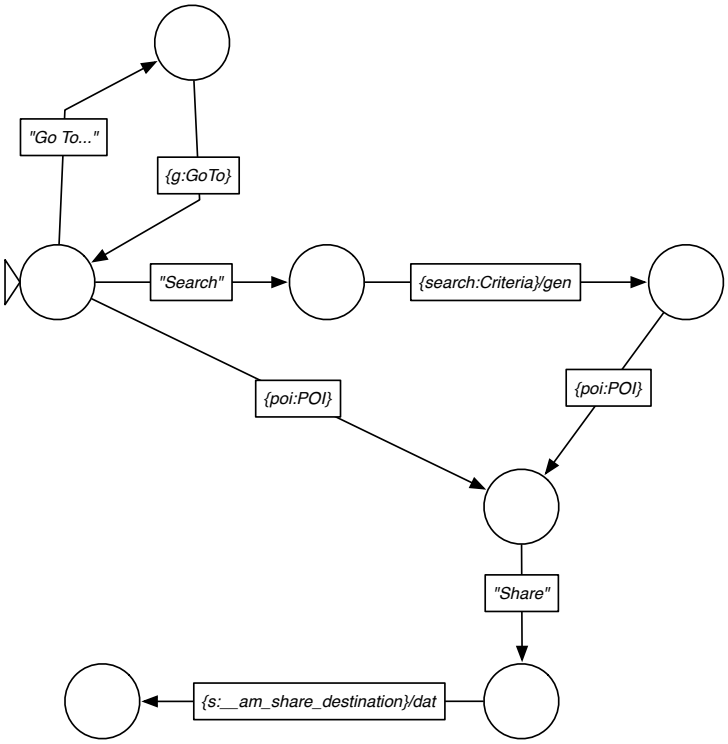


Figure C.15: Venue access application map

was a huge practical benefit" and that in hindsight the modelling of applications this way, with objects and the "semantic" modelling of relationships between them "seems obvious, once you get used to the idea, yet it is not something I have encountered before." He felt that the case system took little to no actual learning but did take some "getting used to": once he had got a feel for the concept of case the actual meanings of the cases were entirely natural.

Participant 5 could not think of any mobile platforms where the model of case would not work. He was unsure how well it would integrate with larger and more complicated data models than that of his prototype, but considered that it would certainly be worth attempting. He thought that case would be useful except where "the interaction with the user was very small", and cited text messaging as an example of an application domain where case would offer little or no benefit. He also said that future work in his specific application would need to centre around "timelines of user tasks (in this case directions in a navigation application)" and was not sure how well this would interact with case. Both after discussing the complexity of data models and the timelines of user tasks, he stressed that he would be keen to evaluate case in these situations, but that the prototype he had built had given him no basis to make statements about case's usefulness for them.

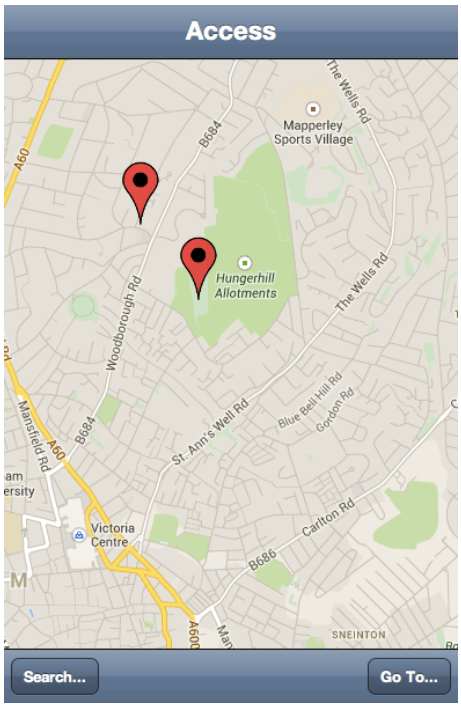


Figure C.16: Venue access: initial map

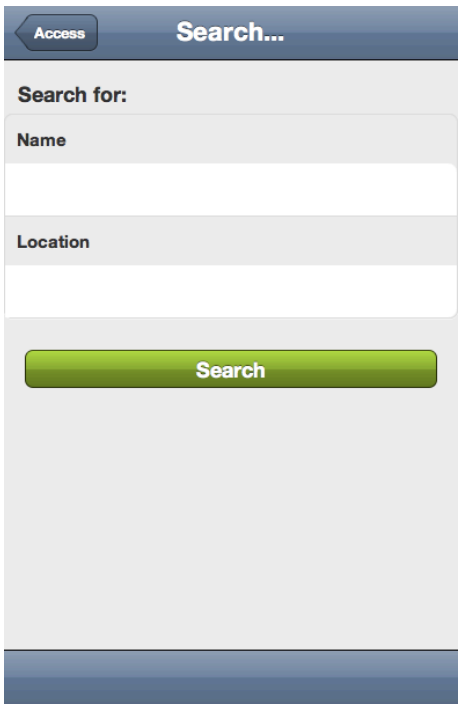


Figure C.17: Venue access: search facility

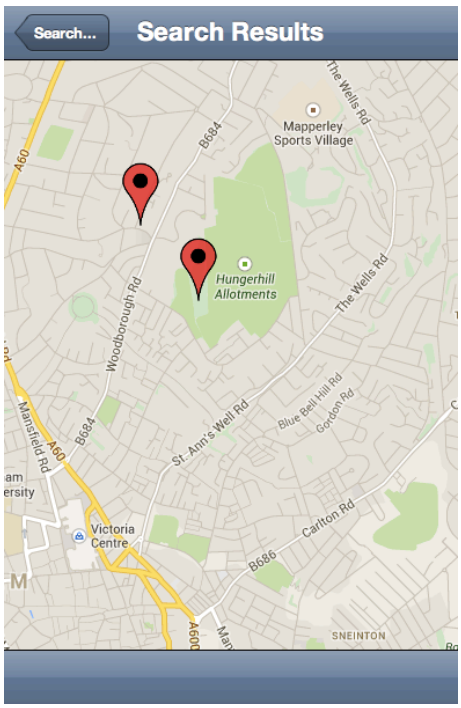


Figure C.18: Venue access: search results

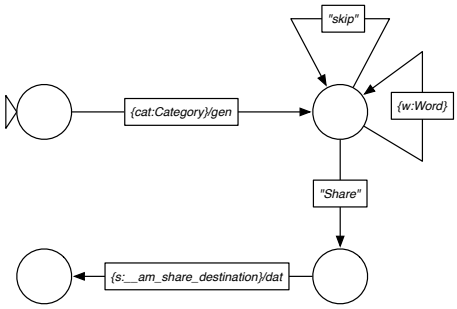


Figure C.19: Language flashcard application map

C.6 Participant 6

Participant 6's application was a simple flashcard game for language learners. He himself was trying to learn both Church Latin and Polish, and used the same application with two different data sets to produce applications for his own use. The purpose of these applications was to allow him and other learners to revise and learn vocabulary quickly while on buses or trains or similar. It therefore had very few states and few features.

The application map for the quiz application is given in figure C.19. When the user starts the application, they are required to choose a category of words. Once they've done that, they find themselves presented with a word, and they can either choose an answer or to skip the word. They can also share a word: this is for a learning method where for each word they get, they should attempt to use it in a sentence on twitter.

Participant 6 used the genitive in a categorial sense, to delimit categories of words. The dative was used for social media applications. He remarked, while building the map, that these were "textbook uses of the cases, just like in the manual". He also noted that as a learner of two Indo-European languages with case systems, the cases in use in the AppMaps case system seemed to be consistent with analogous cases in Latin and Polish as he understood them.

Participant 6 created the application map in two stages: first, he created the map for the ideal phone. Then he added the cases to the map. After this, he designed and created the views themselves.

None of the views in this application performed any plasticity themselves at all. An application-specific stylesheet was used to ensure that the flashcards remained visually centred on larger screens. No logical or dialogue plasticity was used except for the cases on the edges.

Participant 6 was happy with the quality of the user interface of the application for his own use but stated that the physical presentation was not "professional" enough that he would want to release it for other people's use: he said that HTML5 applications—not just AppMaps ones—tended to look "unfinished". He thought that both the results of the genitive and the dative were good quality within this constraint. He had found during development that there was a time advantage in using case, and that it came because he could "think about the tool being on phone or tablet once at the beginning", deal with the plasticity then, and then "as long as I stuck to the right coding style" he didn't have to think about it while writing the code itself. He was of the impression that this was easily enough of a benefit to justify learning the case system.

He could not think of any mobile platform that the case system would be inappropriate for; he himself ran it on Android, but found the iOS versions perfectly palatable. He also tried using the experimental Windows Phone support outlined in chapter 6 and felt that the model worked well on that platform also. He had doubts as to the wider applicability of case to games, which "are often about control and directly interacting with things" rather than data sets. He also mentioned that his application was "probably the simplest data set that case is any use with", because it only had "two layers: categories and the things in them".

C.7 Participant 7

Participant 7 designed a mobile website rather than a mobile application: it was designed to be fetched dynamically from the web and display live information, rather than to be installed on a device. It was designed for an e-Commerce company selling parts for Volkswagen Corrado cars. Participant 7 had not finished building the site by the time the study ended.

The application map for this application as sent to the author is sent in figure C.20. When users go to the site, they have a number of choices. "About us" and "Contact" take the user to static pages with information about the company selling the parts. They can also select one of "Interior" and "Exterior" to select categories of parts or "view basket" to view the parts currently in their shopping basket. From the list of interior or exterior parts, they can select a part, view its details, and add it to their basket or share it on Twitter. From the shopping basket view, they can check out and pay.

In this application, the genitive is used categorially. The "Exterior" and "Interior" edges defined categories of part based on whether those parts were for the exterior or the interior of the car. Participant 7 said that the "view basket" edge was not marked as a genitive because the basket did not constitute purely a category of parts: the page that that edge led to was distinct from the lists of parts and would let the user see shipping pricing and other information in addition to the contents of their basket. The dative is used much as it is in the example projects, to share to social media.

Participant 7 created the map in two phases: in the first, he created the map without cases. He then created the views to fill in the "ideal phone" version of the application, and then iterated through several iterations of how the cases should be assigned to the edges until he found one that gave the meaning that he felt the application should have. In the first such variant, for example, he added only a dative onto the twitter share edge; after this he added the genitive in a partitive meaning onto the "Part" selection edges, but decided that that did not match the "conceptual model" he was trying to put across, and finally settled on adding categorial genitives to the "Exterior" and "Interior" edges. The parts of the interface where the genitive and dative were used did fulfil the relevant requirements of the Android and iOS user interface guidelines.

Participant 7 felt that they were on track to creating a "quality looking application", although simple in functional terms. They had some reservations about the ability of Sencha Touch to create the views that they wanted, but the parts of the application that involved case were of good enough quality for them. He felt that it had made the development of the system faster: it "allowed me to avoid writing boilerplate code for targeting specific device viewports" and "allowed refactoring

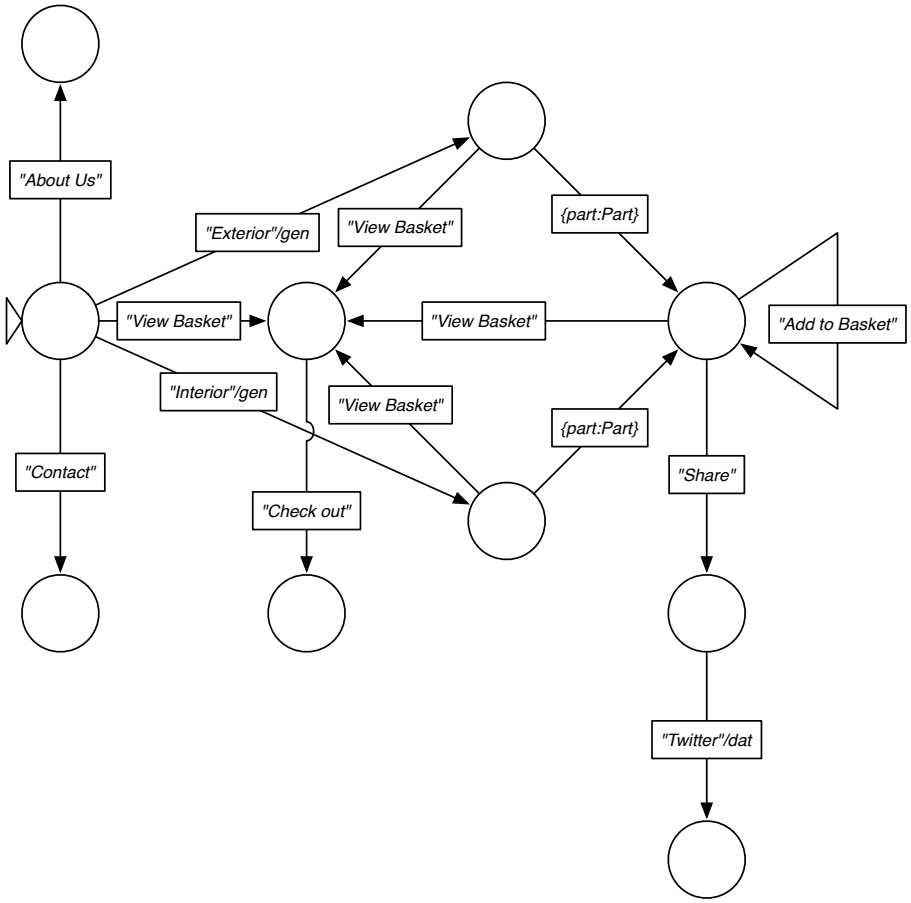


Figure C.20: Corrado parts application map

or making changes to interfaces very simpl[y], which is extremely useful for quickly prototyping ideas". He said that the case system would certainly be worth learning for these reasons, especially the latter: "in a client situation, it would allow you to make changes [to the plasticity] from feedback very quickly, which would greatly reduce the time associated with making those changes." He could not think of any mobile platforms that the model would not apply to. He said that the model was certainly well-suited to simple e-commerce applications which consisted largely of "functionality such as filtering products (gen case) and sharing products (dat case) [and] these [cases] worked very effectively".

"Time hath endless rarities, and shows of all varieties; which reveals old things in heaven, makes new discoveries in earth, and even earth itself a discovery. That great antiquity America lay buried for thousands of years, and a large part of the earth is still in the urn unto us.'

—Sir Thomas Browne, *Hydriotaphia,* I