TOOLS USA '95
Tutorial Presentation

'BuildingBuilding Models of Multimedia Systems: the idea of maximising re-use with Eiffel'

Peter Ward

IMP University of Leeds UK

Tutorial Outline

This tutorial will present practical experience of using the Eiffel OOPL in the development and evolution of working prototypes. The focus will be a critique and evaluation of the 'PARIS' prototype, a software development case study and the latest in a series of prototypes which has included the 'GPE', 'Media Language' and 'GARDEN'. Meyer mentions specific features which make Eiffel well suited to software development and re-use. The possibilities for the effective use of Eiffel facilities will be examined with reference to the real working 'PARIS' project and to the theoretical commercial benefits of re-use and the construction of user-oriented functional interfaces and tools for the organisation, distribution and access to multimedia information. The tutorial will be aimed at software developers, academics, project managers, application developers and marketing personnel. A pragmatic development methodology for real information product will be presented from an information systems perspective. There will be a discussion of the design strategy for the 'PARIS Model, together with any specific metrics of software engineering, including lines of code re-used and hours of programming time saved) in the 'PARIS' project. The use of tools and components other than Eiffel 3 will be described in the 'PARIS' project. The pros and cons of using Eiffel in real world small budget projects will be discussed.

Case Studies, Methodologies and Modelling

A variety of real working models will be shown. Including notions of 'information modelling' and the design of generic, flexible, enhanceable and distributable object-oriented systems working closely with users and in real world applications; high quality, configurable and flexible user-friendly interfaces providing a variety of access and supporting key tasks in information modelling and communication - proportional to functionality, user-friendly and hiding the underlying mechanisms from the user and not over-engineered (what you get is what you need).

Structure of Tutorial

The Tutorial will be in three parts -

*please note a re-ordering of the parts

Part One "Introduction to Multimedia Information Systems Modelling";

Part Two "Engineering Object-Oriented Components: 'GPE', 'Media Language', 'GARDEN' and 'Paris'

Part Three "The Information Systems Perspective; Paradigms and Lessons"

TOOLS USA '95

Paper with reference to Tutorial Presentation

'Building Models of Multimedia Systems: the idea of maximising re-use with Eiffel'Building

Peter Ward
IMP University of Leeds UK

Abstract
The focus of the work to be discussed is the 'Paris Project' - which has been an exercise in 'rapid application development' - originally started in Eiffel and coming to employ Delphi, a new programming tool for application development (with a focus on building and delivering multimedia applications) in the Windows environment. The idea was to continue an evolutionary path of user-led information modelling tools development which started 5 years ago with C++ in the GPE Project, and was followed in 1992 with the Eiffel OOPL used in the Media Language and GARDEN Projects. This series of developments has been presented in terms of a paradigm for building models of multimedia systems. The latest 'Paris Project' has aimed to translate some features of the workstation Unix/X11-based tools into the development of a similar tool for the PC-based Windows platform. The aim was to include and reimplement in a 'multimedia (document) viewer' a number of key interface features - such as the 'history strip' and 'chunking' in Media Language, and the 'pages' displayable within the hierarchical browsable, index-linked and reconfigurable frameworks in GARDEN. One of our ideas was to explore the reality of maximising re-use with Eiffel in the case of a port to Windows. Basic questions addressed at the outset e.g. the extent to which code and design re-use would turn out to be real and finding an effective means to get a handle on Windows, will be compared with the pragmatic solutions to problems encountered, with a report and demonstration of a working model. This technology development will be considered within the context of the search for simplicity, elegance and emergence of working models in a new generation of information systems.

1. Introduction
The tutorial presentation summarised in this paper brings together applied research and development work over 7 years and is based on practical experience gained from the user-led development and evolution of a series of working prototypes, culminating with the 'Paris' information browsing and authoring tool project.

The material is aimed at a wide scope of audience - software developers, academics, project managers, application designers and marketing personnel. A down-to-earth development approach for real-world information products will be presented. We also look at practical and commercial aspects of software re-use, with specific reference to the Eiffel O-O Programming Language.

In the first part of this paper, we present an introduction to Multimedia Information Systems modelling. TodayÕs hardware has the potential for rich multimedia presentation. What kind of software components can do justice to the hardware? How do we identify them, and how do they differ from objects, modules or components in other kinds of applications?

An evolutionary path of user-led information modelling tools development will be illustrated. Each successive software project at Leeds-IMP has built upon the design of the previous one. The central theme has been information presentation and management (Ward 1994). In a programme of applied research started 5 years ago, C++ (Glockenspiel) was employed in the GPE Project (Parrott

and Ward 1991), and this was followed with the employment of the Eiffel OOPL (ISE Eiffel Version 2.3) which was used in the Media Language and GARDEN Projects (Parrott and Ward 1993; Ward and Parrott 1993; (Howard 1994) . This series of developments has been presented in terms of a paradigm for building models of multimedia systems (TOOLS Europe95). MediaLanguage has a sophisticated "chunking" screen model, and required the application developer to write a program in Media Language to define the information content. GARDEN provided an alternative full-screen model, dropped the programming (scripting) requirement, and emphasised information presentation and access and the construction of frameworks. Two Eiffel Class libraries are at the core of these tools. The X11 Class library provides a simple but powerful interface to the X11 display system and is available for ISE Eiffel 2.3.4. The Unix Class library provides access to the underlying features of a Unix process, file/directory paths and network sockets. The full library is available for ISE Eiffel 2.3.4 and the TowerEiffel compiler. The Tower code should also work with the Eiffel/S compiler. The Unix process abstraction with a command-line argument processor is available for the
ISE Eiffel 3.2.4 compiler.

The latest 'Paris Project' has aimed to translate some features of the workstation Unix/X11-based tools into the development of a similar tool for the PC-based Windows platform. The aim was to include and reimplement in a 'multimedia (document) viewer' a number of key interface features - such as the 'history strip' and 'chunking' in Media Language, and the 'pages' displayable within the hierarchical browsable, index-linked and reconfigurable frameworks in GARDEN.
The 'Paris Project' is taking this technology to the Windows mass-audience, and is a means of providing advanced information presentation and management
based on the HTML document format as found on the world-wide-web. None of these tools obsoletes the others. They occupy distinct niches and appeal to different categories of user. Yet the common thread of "information presentation and management" runs through them and opens up wide possibilities for re-use.

The information modelling process must define generic, flexible and enhanceable object-oriented systems which lead to high-quality, configurable, user-friendly applications. Key factors to achieve this include hiding the underlying mechanisms and avoiding over-engineering (what you get is what you need).

In the second part of the paper we look at real-world aspects of software re-use. Meyer has identified specific features of Eiffel which make it well suited to software development and re-use (Meyer 1988, 1990, 1992). These features are examined with reference to how they affected our projects. With 'Paris Project' we decided to take an active approach to reuse, and the outcome of this is described here. An alternative strategy to using Eiffel is discussed in which other tools and components - principally Delphi (an object-Pascal "rapid application development" environment) are employed, with a discussion of the relative merits of the contrasting methods.

In the light of experience with a real time small scale project, we will critically examine how an object-oriented approach and the use of object-oriented tools is a real advantage. Our first idea was to report our experience in terms of specific metrics (including lines of code re-used and hours of programming-time saved). We also intended to look at less-easily-measured items such as product quality and time-to-market, and attempt to quantify the value of any repositories of reusable software which are being produced. We will critically examine whether re-use has really occurred and whether it is possible to realise the benefits it promises. A checklist of tips needed to ensure that the potential benefits can be maximised is presented.

2. The 'Paris Project' - an opportunity for maximising software re-use?
The working 'Paris Viewer' model has been developed in 6 months with a focus on a simple stand-

alone capacity to handle HTML standard material and provide a very simple interface and access to multimedia materials. Material from 'Eiffel The Language', 'Alice in Wonderland' and 'Postman Pat' will be illustrated, along with a number of Media Language and GARDEN case studies. The first experimental object-oriented information modelling tool - the GPE - will also be illustrated.

[Place Pictures Here

GPE x 1 or 2

ML x 1 (ml2) (Weaving) or 2 (ml3) (VUIS)

GARDEN x 1 (good old EFC) or 2 (Access or Amsterdam Class Libraries)]

2.1 Code Reuse

Programmers often think of reuse in terms of "reusing lines of code". In this regard, we were at somewhat of a disadvantage. MediaLanguage and Garden were written in Eiffel 2.3, and it was planned to write 'Paris' in the latest version of Eiffel. There are substantial differences between Eiffel 2.3 and Eiffel 3, and it would not have been possible to reuse any code without conversion. In this case, we needed to consider wider reuse issues than simply "reusing lines of code". Environmentalists have a slogan: "Reduce ... Reuse ... Recycle" - in that order of preference. This same slogan makes a lot of sense for software development.

2.1.1 "Reduce"

The first step, which is often overlooked, is to reduce the amount of code which must be written in the first place. This is a design-time task that requires great discipline. It is hard to resist the onslaught of creeping featurism, and the larger the design team - the worse the problem.

A careful design can vastly reduce the size of a project, whilst providing a more elegant, more smoothly functioning deliverable. As well as a big reduction in development time and cost, the finished software is likely to run faster, require less storage, install more easily, contain fewer bugs and require less support.

For the 'Paris Project', we considered our target end-user audience carefully. We deliberately avoided trying to copy the feature set of other loosely-related tools. To do so would not only have consigned us to forever playing "catch-up", but it would actually have made the product less usable in our target market.

We also carefully identified a subset of features that were appropriate to the entry-level version of the tool, with further features to be added to later versions. This allowed us the possibility of getting a first working model as a product onto the market very quickly, and to keep it on the market as an entry-level doorway after other products in the family are released.

Our first commercially-viable release contains perhaps 10% of the code of its competitors, yet contains all of the functionality that many of our users will require.

Similarly, we have reduced the design effort by freely adapting existing standards wherever possible. Whereas Media Language and GARDEN used proprietary media modelling formats, we use standard HTML with a few trivial extensions to support the 'Paris Project', thereby avoiding the need to design our own formats, and opening up opportunities to reuse other people's HTML software components.

2.1.2 "Reuse"

Here we refer to the re-use of existing software, without modification but possibly after adaption. We see two main categories of such re-use: tapping in to components already present at runtime, and adapting existing components at compile time.

Reusing components at runtime

The 'Paris Project' was designed to be suitable for the mass-market distribution of multimedia documents, for example by means of CD-ROMs supplied with books. For this application, Microsoft Windows is the common denominator and we had no need to provide for cross-platform compatibility with Unix or any other operating system. This has made it possible to aggressively re-use software which is accessible at runtime -- in particular the Microsoft Windows API. One should not think of this API as merely a graphics interface. It also provides facilities for keyboard-handling, decompression, sound, video, installation, configuration management, file and stream I/O, printing, communications, memory management, palette control, standard data formats, timing etc.

By reusing the available routines wherever possible in the construction of the 'Paris Viewer', we have greatly reduced the size of our development task. There is a slight price to pay in terms of conceptual clarity. The available routines are a 'hotch-potch' built up over the years. There is no consistent design behind them, although recent versions are much improved. This is a small price to pay compared to the availability of over a thousand routines, most of which are useful and reliable.

Perhaps some would claim that we have paid too high a price by losing cross-platform compatibility. Although we have no need for this feature today, might not the future be different? It is not always possible to cater for cross-platform compatibility when the focus is on delivering functionality in the short-term. In the case of the 'Paris Project', we are not concerned, because it seems that any future target platform is likely to support MS-Windows applications at a binary level or by emulation, enabling us to continue to develop just one version.

Other run-time reuse can be obtained by exploiting shared components in the form of dynamic link libraries or OLE-2 objects. We have not made use of these in the entry version of the 'Paris Viewer', although we will look for opportunities to do so in more advanced versions.

Adapting components at compile time

When people talk about software reuse, often they are referring to the adaptation of existing software at compile time. For example, by inheriting from an existing software component we can adapt and reuse previously-developed code. With a good collection of software components, most of an application can be written by simply tying together existing components.

Eiffel has been designed from the start to be a language for writing libraries of reusable components, and seems likely to fulfil this promise. However, at the time of writing we did not have access to any suitable libraries other than those which are supplied with the compiler. In particular, our existing libraries from Media Language and GARDEN were written in a previous version of Eiffel and could not be inherited from without modification.

Design reuse

A further aspect of reuse does not relate to code at all, but to other aspects of the underlying system design which can be gleaned from the code.

For example, GARDEN supported a carefully-tailored colour palette specifically optimised for multimedia applications. Although we did not use that palette unchanged, we derived our base palette for the 'Paris Project' much faster than if we had designed it from a blank slate.

Our reference source for the GARDEN palette was not the Garden documentation (which gave a good general overview of the colour system), but rather the GARDEN code which gave an unambiguous specification of it through its implementation.

2.1.3 "Recycle"

This leads us to the third part of the environmentalist's slogan: "Recycle". Environmentalists know that recycling is of marginal economic and environmental benefit, and is only desirable by comparison to disposal. Reduction and reuse are much more worthwhile. Again, a parallel can be drawn with software development.

Code recycling seemed to be the only way we could get much benefit from our existing code, written in Eiffel 2.3 under Unix. We critically examined the existing code, and came to the conclusion that 10% of the programming requirements for Project Paris could be met by a direct conversion of existing code to the latest version of Eiffel. We estimated that it would take us half as long to convert and test existing code as it would have taken to write and test completely new code.

Therefore, we expected to save approximately 5% of the programming time and cost of our project by means of this "code scavenging".

Prior to the ascendancy of object-oriented languages, "code scavenging" was often the only readily available technique to achieve reuse. Every self-respecting programmer in the sixties had their card deck of reusable Fortran or Cobol routines!

Although object-oriented languages allow reuse at a much higher level of abstraction, they do not render code scavenging obsolete where no better technique is applicable.

3. Our latest experience

In the latest phase of the evolutionary series - the 'Paris Project' - we had to change our plans suddenly in April this year when it became apparent that due to vendor delays there was not yet a suitable Eiffel compiler under Windows. We decided, reluctantly, to switch to Delphi. The change proved very smooth for various reasons.

First, we had actively pared our design down to its central essence, so the scale of our work was much reduced regardless of development language. Second, Delphi proved to be a highly-functional and efficient development environment combined with an effective programming language. Third, Delphi is about as close as we could hope to get to Eiffel in another language, particularly after Borland's recent changes which bring a number of Eiffel-like features to the language. For example, Delphi objects are references with automatic and transparent dereferencing, just as in Eiffel. This third reason enabled us to carry on with "code scavenging" to about the same degree as we had originally planned.

Although Delphi's programming language does not offer all of the benefits of Eiffel (in particular, it lacks true genericity, multiple inheritance, assertions and garbage collection), it did provide us with access to the Delphi Visual Component Library which we used extensively. These very useful components are heavily parameterised and may be adapted interactively at design time.

4. Conclusion

Reuse is a multi-faceted aspect of software development. One should look everywhere for reuse opportunities, because there are many ways to gain its benefits.

In terms of the evolution of the series of "information modelling tools" - the 'GPE' is classical

object-orientation. Very like a visual smalltalk with a little compile-time static typing thrown in for good measure. GARDEN, with its in-built framework, provided for 'information modelling' in terms of frameworks - the framework is information modelling. The rest is 'model visualisation'. Both are necessary to really understand a problem and provide a mechanism.

Object-orientation is an enabling technology, it allows good developers to achieve new levels of productivity (Rambaugh 1991). Poor developers will still be poor, hiding behind bad practice and dubious processes.

Essentially with the tools - Media Language and GARDEN - there has been a reuse of ideas but little direct code reuse. This is 'developer reuse' and is the commonest form of reuse. The same would have happened if the programs had been written entirely in C, LISP, COBOL or Ada. Key ideas from these tools have been reimplemented in the construction of the 'Paris Viewer' and the embedded framework. The earlier tools have enabled the manipulation of 'display objects' which were only superficially 'object-oriented' in the usual sense of the term (GPE is the exception).

Eiffel defines precisely what is an 'argument' and what is a 'parameter' (with an argument attached to a class feature whereas a parameter is attached to a generic class). In other languages these two words are used interchangeably. The clear separation aids clarity enormously. Eiffel is a small yet powerful OOPL - providing two very important and basic features: assertions and simple relationships (client-server and descendent ancestor). However, Eiffel has disadvantages too. It is a single executable, and its memory requirements are considerable - memory is never returned to the operating system. Although the working set does reduce as the garbage collector does it's work, our experience is that _really_ using X and the typical Eiffel development environment require at least 24Mb of memory and preferably 32Mb. Otherwise the compiler causes intolerable swapping and reduces processor utilisation to less than 50%. This is no good for any sort of rapid development. Unfortunately this is the price to pay for a powerful language with a monolithic compiler/development environment.

Under Windows the problems are likely to be exacerbated. Windows is not renowned for it's excellent memory management. Real memory above 16Mb is not used effectively and the paging algorithm is primitive. The sheer size of the typical Eiffel system, if used in anger, tends to bring Windows to it's knees.

A fundamental question that we faced at the outset of the 'Paris Project' was 'which is more important the product or the method'? The real challenge to the developer is to identify best way to get a product out in a rapidly shrinking window of opportunity. It was depressing in the event not being able to use Eiffel, but that doesn't mean that other tools e.g. C++ or Delphi cannot be used in the Eiffel style. Important constraints appeared to be restriction to protected and public, only using single inheritance, and trying hard not to miss assertions. Undoubtedly, the 'Paris Project' benefited significantly from the "Eiffel mind-set" and the application, for example, of pre-conditions in Delphi which trapped a large number of errors in the main program.

In the 'Paris Project', the construction of a hypermedia application is viewed as requiring elements from three arenas:

(i) A viewer. This can be thought of as the mechanism by which the user interacts with the application and by navigating the framework, accesses the content. An interaction should result in information being transmitted from the document to the user in a structured and meaningful way (for that user). The way the viewer is built and the interface it offers the user will determine how easy it is for a particular user to access the information they require. The 'Paris Viewer' draws upon the research in the area of Human-Computer Interaction, Object Oriented components and

Ergonomics so that its implementation will be attractive and engaging to a human user.

(ii) The content. These can be thought of as the basic building blocks of a hypermedia application, text, sound, video, graphics and other data types. Style guides for good, clear prose; sound orchestration; graphic design and filming exist and are widely used in the media industry. Obtaining good 'stuff' as your content is the first step on the road to a high quality hypermedia application.

(iii) The framework. This can be thought of as the 'web' upon which the content is woven. The framework determines the boundaries and the paths a user can take through the application and also the paths a user may *not* take. The publishing industry has methods of editing a book or a magazine; educational and training material must be organised to make it more navigable and possibly easier to understand. Essentially though, this is largely still working on a two-dimensional surface. A hypermedia framework adds a third dimension and a whole level of complexity (Gronebaek 1994; Halasz and Schwartz 1990).

Over the past six years IMP has been conducting an ongoing program of applied research in the area of hypermedia applications and we have built many examples. The focus has mainly been on the 'viewer/reader' end of the above triumvirate, although with the GARDEN project inroads were made onto the framework area.

The emergence of the world-wide web and the advent of global hypermedia has opened up huge amounts of content to the mass consumer audience. As anybody who has done even a small amount of 'net surfing' will have realised, it is desperately easy to get lost or travel endlessly around in circles in the tangled mesh of links liberally scattered over the web's content. It is clear that the 'framework' needs looking at if the world-wide-web is to fulfil its potential of providing easy access to information. The questions 'Should I make a link?' and 'How should I make a link?' need addressing.

The programme of applied research being conducted by the IMP Group continues to seek answers to these questions and to produce better, higher quality and more accessible hypermedia applications.

Among the issues raised in the tutorial and summarised in the paper are: user attraction, ease-of-use and the engagement of the end-user; interactivity, preference and performance; the nature of information modelling frameworks and mechanisms for the organisation, cross-referencing and interactive presentation of multimedia information; the dangers of free-association and getting lost in 'hyperspace'; and using OOT to achieve realistic results in realistic time-frames - to construct models, frameworks and components, with a potential for enhancement in response to need and logical evolution an inherent feature of design.

Because multimedia is a relatively new development, many potential applications have yet to be exploited. The capabilities of the "objects" in our applications often do not become apparent until the design phase. The analysis phase is scaled down, because there is not yet a real-world model for many aspects of what we are doing!

The basic aim of the programme of applied research and development aim has been to build information modelling tools and better mechanisms for accessing and distributing multimedia information. Eiffel has proved a serious tool in building operational systems in telecommunications (Osmond 1994), banking and multimedia systems (Parrott 1993). However, in the 'Paris Project' - a rapid application project with the aim of delivering key simple functionality onto the Windows platform - another object-oriented tool Delphi provided more immediate leverage.

The Eiffel OOPL presents the possibility of supporting all the stages of development - from analysis to implementation. The language provides a number of key features and great potential and yet is it in reality the best tool for building models of multimedia systems? Is it a question of the "perfect being the enemy of the good"?

5. The future

Reuse never stops with the current project. The most effective reuse is available when it has been planned for, as in the case of a library which has been designed from the outset to support reuse.

We looked toward the future in two different ways with the 'Paris Project'. First, we expect to produce software which can be generalised into Delphi Visual Components for the benefit of future Delphi programmers. Second, we have adopted a coding style within Delphi which closely mimics our preferred Eiffel style. This has been done with a view to a possible future conversion of our software to Eiffel.

Acknowledgements

The author is pleased to acknowledge Colin Parrott for the work - software design and engineering - on the GPE, Media Language and GARDEN, to Graham Lovell of Sun Microsystems for the support of the Unix/X11 work and to Jayne Gray for her 'lamp lighting' when the going gets tough.

References

Gronbaek K., Hem J.A., Madsen O.L. and Sloth L "Hypermedia Systems: A Dexter-based architecture" Comm. ACM Special Section: Hypermedia 37 (2) 65-74 (1994)

Halasz F. and M. Schwartz "The Dexter hypertext reference model" In Proc. Hypertext Standardisation Workshop (Gaithersburg, Md) 95-133 June (1990)

Howard R. "Eiffel Technology Underlies Multimedia Courseware Development Tools" Object Magazine 3 (6) 36-41 (1994)

Meyer, B "Object-Oriented Software Construction" Prentice-Hall (1988)

Meyer, B "Lessons from the Design of the Eiffel Libraries" Comm. ACM 33 (9) (1990)

Meyer B "Eiffel: The Language" Prentice Hall (1992)

Osmond R. "Experience from a large Object-Oriented Project" TOOLS Europe '94 Technology of Object-Oriented Languages and Systems, Versailles Prentice Hall (1994)

Parrott C and Ward P S "The GPE: Graphical Programming Environment - An Object Oriented Information Modelling Tool". TOOLS Europe '91 Technology of Object-Oriented Languages and Systems, Versailles, France Prentice Hall (1991)

Parrott C and Ward PS "Media Language: A Rapid Application Development Tool using Eiffel X11" Proc. of TOOLS USA '93 Technology of Object-Oriented Languages and Systems, Santa Barbara. Prentice Hall (1993)

Parrott C, Ward PS and Arshad FN "Project Feature: Media Language" in Eiffel World 3 (1) Fall (1993)

Rambaugh J., M.Blaha, W. Premerlani, F. Eddy, and W. Lorenson "Object-Oriented Modelling and Design" Prentice Hall Englewood Cliffs, NJ (1991)

Ward P.S. Guest Editor for Current Psychology: Research and Reviews Special Edition "Hypermedia and Artificial Intelligence" 9 (2) Summer (1990)

Ward P.S., and C. Parrott "GARDEN: an environment for the organisation and distribution of multiuser multimedia applications on the network" Invited Seminar SUN Microsystems Mt.View September (1993)

Ward PS 'Distributed Digital Multimedia Materials : towards new document centred user interfaces?' Interactive Media International, Summer (1994)