

University of Edinburgh

Division of Informatics

Migration on Request

4th Year Project Report
Computer Science

Adam Rusbridge

May 28, 2003

Abstract: Digital Preservation is defined as the act of ensuring the longevity of digital objects. Migration on Request is a method used to preserve these objects. This report presents an overview of the issues faced in the digital preservation field and extensively discusses the implementation of a tool and the corresponding issues highlighted.

Acknowledgements

Thanks go to my project supervisor, Najla Semple, for her endless patience, encouragement and guidance throughout the year.

Thanks must also be given to the assistance from Paul Wheatley, Phil Mellor and Derek Sergeant of the CAMiLEON team. Their advice was invaluable when getting the implementation off the ground. Eric McKenzie provided very useful comments during the interim meetings.

Finally, thanks to friends and family for humoring me during the more stressful periods of the year.

Contents

1	Project Summary	1
1.1	Introduction	1
1.2	Objectives	1
1.3	Conclusions	1
2	Introduction	3
2.1	Background	3
2.2	Archives	4
2.3	Significant Properties	4
2.3.1	Determining the Designated Community	6
2.4	Software Dependence	6
2.5	Summary	7
3	Current Solutions	9
3.0.1	Technology Conservation	9
3.0.2	Printing to Paper	9
3.0.3	Encapsulation	9
3.0.4	Virtual Machine Software	10
3.1	Emulation	10
3.2	Migration	11
3.3	Migration On Request	11
3.4	Longevity Of Solutions	13
3.5	Summary	14
4	Authenticity	15
4.1	Trust	15
4.2	Archival Stages	16
4.2.1	Before Ingest	16
4.2.2	During Archival Custody	18
4.2.3	Upon Retrieval	18
4.3	Summary	20
5	Design	21
5.1	Objectives	21
5.2	Big and Little Endian	22
5.3	Raster Formats	22
5.3.1	Input Formats	23
5.3.2	Output Formats	26
5.4	Significant Properties of Raster Formats	28

5.5	Languages	28
5.5.1	C and C++	29
5.5.2	Java	29
5.6	Development Considerations	31
5.7	Intermediate Structure	31
5.8	Unsupported Attributes	33
5.9	Number Systems	34
5.10	Summary	35
6	Implementation	37
6.1	Implementation of Structure	37
6.2	The Value Class	37
6.3	Conversion between Unsupported Elements	38
6.3.1	Continuous to Paletted	38
6.3.2	Colour Depth	39
6.3.3	Bitonal/Greyscale/RGB/CMYK	40
6.3.4	Transparency	40
6.3.5	Pixel Resolution	41
6.3.6	Compression	41
6.4	Input	41
6.4.1	PNM	42
6.5	Output	42
6.5.1	PNM	42
6.5.2	PNG	43
6.6	Implementation Restrictions	43
6.7	Summary	44
7	Testing	45
7.1	Verification	45
7.2	Validation	46
7.2.1	Reversible Migration	46
7.3	Robustness	47
7.3.1	Multiple Migrations	48
7.4	Summary	48
8	Evaluation	49
8.1	Research Considerations	49
8.2	Evaluation of Intermediate Structure	49
8.2.1	Comparison with Java Image Class	49
8.2.2	Duplication	51
8.2.3	Range of Features	51
8.2.4	Number Systems	52
8.2.5	Value Class	52

8.2.6	Big and Little Endian	52
8.3	Portability and Longevity	53
8.3.1	Coding Style and Comments	53
8.3.2	Java--	53
8.4	Implementation	54
8.4.1	Programming Experience	54
8.4.2	Resource Utilisation	54
8.4.3	Number System Issues	55
8.4.4	Implementation Output	55
8.5	Summary	56
9	Conclusions	57
9.1	Original Evidence	57
9.2	File Formats	57
9.3	Intermediate Structure Design	58
9.4	Languages	58
9.5	Authenticity	59
9.6	Reversability	59
9.7	Future Developments	59
9.8	Conclusion	59
	Bibliography	61

1. Project Summary

1.1 Introduction

This document presents an overview of the motivations, authenticity issues and solutions proposed by the Digital Preservation community, followed by a detailed look at the technical issues faced in the development of a raster graphic Migration on Request tool. Although there are many raster graphics converters available, converters for fields such as Computer Aided Design have not been addressed. While these are beyond the scope of this project, the level of difficulty in this example will allow me to demonstrate, within the project timescale, the general issues an implementation must overcome.

1.2 Objectives

There are two distinct objectives that I have laid out within this project, requiring different skills and backgrounds.

- Investigate the techniques and issues involved in the field of digital preservation.
- Develop and implement a raster graphic Migration on Request tool, highlighting the design and implementation issues involved.

The quantity of literature available on digital preservation will require significant research and reading; completing this first objective will allow me to develop and present my own opinions and ideas on the topics.

The second objective is a combination of several problems. It will require research gathering on the chosen formats. The technique demands a careful, well planned design to be successful. Designing and implementing the tool will use many of the computer science and software engineering techniques I have learnt and will test my software programming ability.

1.3 Conclusions

The first objective was completed satisfactorily and the first few chapters of this document serve as an introduction to the range of issues that are involved within digital preservation.

At the end of the project, a limited implementation of the tool was completed. The open-ended nature of the tool make this understandable. A detailed evaluation of the technical problems encountered has been given, along with an evaluation of the factors resulting in the restricted implementation. A detailed and revised design has been developed through the document, which should serve as a comprehensive specification for an alternative implementation.

Writing a graphics converter is not a trivial task. Attempting the implementation using an open, extendable methodology required a significant amount of specification research and programming work. When combined with the research of digital preservation required, focusing energy on even slightly unrelated topics made it easy to fall behind. However, despite the implementation issues, the information collected and discovered and both digital preservation and Migration on Request mean this project can be regarded a success.

Whereas I previously knew nothing about Digital Preservation, this document shows the understanding that has been gained in the basics of most topics involved and the detailed issues faced by some specific areas. The work completed on the implementation indicates the design of the tool could result in a successful application. Critical evaluation of the different stages of the project has been conducted, and it has become clear where more energy should have been focused. It is often considered equally important, if not more so, to make mistakes and learn from them as it is to successfully complete a task without any mishaps.

2. Introduction

2.1 Background

Modern society has a reliance on digital methods that is rapidly increasing. New applications are continually being created that allow us to achieve more on a computer; consequently more data is being stored in a digital form. Competition for market niches has caused software vendors to deliberately change formats and hide their proprietary specifications.

Digital preservation is an issue that is becoming both more important and relevant every day. It can be defined as the act of ensuring the longevity of digital objects. A key property of a digital object is that it can either be a document "born digital" or a product of an analogue to digital conversion; however it is only accessible using software and hardware. A digital object contains both the data content and the relevant software to view and execute this. Without the software method in some form, the data is meaningless.

This document is intended to discuss the practical difficulties faced in preserving digital material. The timescale in discussion is in the order of fifty to a hundred years. Rapid advances in technology are currently being made and generations of both hardware and software platforms are being replaced within years of being introduced. The ultimate achievement of a preservation method will be to enable access to an object indefinitely, without altering either our view of it or the intended meaning behind it.

It is not possible any more simply to create a hard-copy of the object in question. Firstly, the sheer quantity of textual documents worth preserving would require masses of paper; converting digital audio and video recordings to analogue drastically reduces the quality. Converting and changing documents in this manner is not creating solutions but merely side-stepping them, and we still have to prevent the decay of the new analogue materials. While some materials (e.g. paper) can be extremely durable, they can also be under serious threat from fire [5].

Secondly and more importantly, many digital objects are now dynamic. For example, databases change their state daily and web pages often contain links to external sites. Guidelines must be published determining the boundaries of the digital object and the extent of the preservation task necessary.

Paul Wheatley of the CAMiLEON [9] project proposed the following requirements a preservation method should adhere to [46].

- Preserve a copy of the digital object.

- Render digital objects and provide the accurate reproduction of the properties which are deemed to be significant.
- Verify the accuracy of the preservation and rendering.
- Provide the first three requirements in the long term.

This is a logical and concise breakdown of the factors involved. This chapter details the stages to preserve a document; The following two chapters discuss methods for the following two requirements. The last requirement is only achieved by careful design and collaboration between the first three.

2.2 Archives

Digital objects can be stored in many places, such as personal computers, servers and removable media. It is clear that long term storage facilities need to be created to hold objects worth preserving. Digital archives are being established to ensure this accessibility is maintained. While standards and requirements for the practical implementation of these are still being established, their purpose is clear. They aim to provide the access reliably and ensure that an object entrusted to them will remain authentic.

Much of the current archival framework is based upon the Open Archival Information System (OAIS), proposed by the CCSDS [16]. This model covers all aspects of an archive, much of it from a management perspective. However, it should be read as a model rather than a design specification- an OAIS can be designed and implemented in many different ways all consistent with the model. Within an archive, an object consists of a number of packages which change as the object passes from one part of the archive to another. The Archival Information Package is preserved and maintained over time. Holdsworth and Sergeant give a detailed introduction to the application of this information when using an implementation of the OAIS model [20].

2.3 Significant Properties

Significant properties are those properties of digital objects that affect their quality, usability, rendering and behaviour [18]. When choosing a preservation method, care should be taken to establish what properties are necessary for the document to serve its intended use. The implication here is that we do not always require an exact replica of the object to be output to preserve the object successfully.

Establishing the significant properties will be a subjective evaluation as different institutions will have very different requirements for an object to satisfy; these cannot be selected by a single set of decision-making rules. Rather, they will depend on institutional priorities and usage and will result in properties that may be different for each preservation. For example, in the field of graphics the preservation of colour is often a significant property, but if we were preserving architectural plans colour may have been used solely for the purpose of aesthetics.

An object's significant properties are neither absolute nor static. The decision on which properties to preserve is ultimately a collection management decision and must fulfil the repository's responsibilities and needs of its stakeholders. Materials core to the collection might retain all the original functionality, whereas more peripheral materials might not include their original full complement. Sometimes we want the content in a modern wrapper to enhance functionality, such as navigation. We need to determine whether this is reasonable.

The significant properties of a digital object dictate the level of detail the underlying technical form requires, and thus the level of documentation required to ensure this detail remains accessible. Naturally, the decision of these properties has serious resource implications: increasing the functionality of an object also requires an increase in the technical information necessary to render this functionality.

We cannot go through objects individually to determine their significant properties, so repositories need to establish policies to apply to different classes of object. Technical metadata detailing the access and rendering functions needs to be determined by system designers and software suppliers, as this is beyond the field of most research libraries and archivists [38].



Figure 2.1: Removing Formatting Can Remove Meaning

Preservation can require migrating to a certain platform or normalising against specific constraints. Since this work can result in the degradation of the document it has become necessary to establish what message the document is intended to convey. For textual documents, a common loss is in the presentation and layout of the document. Often this would not eradicate the message behind a document (although figure 2.1 shows a situation where it can), however it might reduce the fluidity and context of the text. A complex relational database that displays its findings as HTML has many more dependencies that must be considered. The functionality of the database must be retained, along with details of the

method of presentation. Removing these properties may result in an almost incomprehensible object to a user accessing it for the first time.

2.3.1 Determining the Designated Community

Preservation takes place for a designated community. ‘Community’ is a way of saying a group is interested in certain aspects. The bookbinding community is interested in different things than the literary community, which is interested in different things than the simple reader!

The level of preservation required is determined by the information and functionality required by the users. The technical ability of a user influences the automation required by systems. For example, one user may be comfortable with a command line interface whereas another may require a detailed and explicit user interface to perform similar tasks.

The combination of the users technical ability and the level provided to them by access systems is known as a repository’s knowledge base [38]. By understanding the knowledge base, we can estimate a level of detail required in technical metadata to reproduce this functionality.

A digital repository can have many users that will never physically go to the institution, preventing us from explicitly understanding the designated community. However, if we can understand the knowledge base, we can establish the level of functionality required by the community and continue to provide for their needs.

2.4 Software Dependence

One of the difficulties (and saviours) of preservation is that the properties as described above are often abstract from the actual data itself. Storing data as a bytestream means that without the software to reproduce those factors we deem significant the data is meaningless. Careful documentation needs to be stored detailing how information can be extracted from the data.

Standards organisations have been established globally attempting to enable the widespread use of a smaller well supported set of formats, releasing new (group validated) versions periodically. However, software vendors release tailored proprietary formats without releasing specifications to gain the upper hand over the competition. This escalates when other companies do the same, ending in the wide selection of similar formats that can be seen today.

The advantage of this cycle is that developments are rapidly occurring and new features are rapidly added to enhance formats; the disadvantage is that when

companies no longer support older formats and features it becomes difficult for others to produce software to render these formats without the specifications. Since software is always required to translate the digital bits into the form we see, there is the always a risk that “regardless of how well the bits were stored, the document may be inappropriately altered when the stored bits are retrieved and presented for use” [23]. Without the specification of the format to inform us how to retrieve the information, the likelihood of this is increased.

2.5 Summary

This chapter was intended to give the reader an introduction to the issues faced in the digital preservation world. We have been introduced to the archiving model many institutions are basing their designs on and have learnt of the rapid life cycle of software causing the issues faced in this field. Most important from this chapter is the concept of significant properties, which allow the institution to determine which elements of an object they wish to preserve. Once we have established a set of properties we would like to keep, we must find a method enabling this.

3. Current Solutions

Once we have determined what we want to preserve, we need to establish a method to achieve this. There are a number of methods that have been discussed. The following have been suggested by the Digitale Bewaring Project in Migration: Context and Current Status [7] and then further critiqued by CAMiLEON's Paul Wheatley [47]. Note that this is not an exhaustive list of methods of preservation: the use of XML is being widely researched and variants on many of the following methods exist.

3.0.1 Technology Conservation

A short term solution is simply to maintain the hardware environment the software currently runs upon. However, when this technology is superseded the production of older components slows and eventually dies out. Specifications of older items go missing and companies go bust. The skills required to use and maintain the machines become rare. If the physical object we are preserving fails at some component, it can become very difficult to repair. When this occurs, it can become difficult to access a file to move it to a safer environment, before even trying to render it. The CAMiLEON Domesday [8] project faced many difficulties associated with this method.

3.0.2 Printing to Paper

This is not a viable long-term solution. Printing to paper loses much of the functionality of digital objects and only works well for flat objects, such as text documents and still images (often limiting the functionality, such as the loss from printing vector graphics). Databases were simply not designed to be printed out and would result in a single, limiting view of the information. However, desperate for an attractive alternative solution many institutions employ printing as an interim approach.

3.0.3 Encapsulation

The suggestion here is to retain the object in its original form, encapsulated with a set of instructions on how the original should be interpreted. These instructions consist of a “detailed formal description of the file format and what the information means” [7].

The difficulty here lies in determining and verifying the suggested ‘detailed formal description’. It is difficult to determine whether this description is adequate; undocumented software and hardware bugs can pose yet more problems [47]. Currently, this is a deeply theoretical approach as no suitable formal description has been proposed. Jeff Rothenberg’s highly referenced Emulation approach [40] uses a combination of Encapsulation and Emulation; while the technique is advocated by many influential figures, it has many critics as well.

3.0.4 Virtual Machine Software

An approach suggested by Raymond Lorie of IBM is that of Virtual Machine Software [30]. This suggests “writing a program to carry out the interpretation in the machine language of a ‘Universal Virtual Computer’ (UVC). This would be written at the time the record was archived and would be preserved together with the record. To interpret the record in the future, a UVC Interpreter would be required, produced from the specification of the UVC. This approach is similar to that used to achieve interoperability of the Java platform.” [7]

Although this sounds like a promising method it is only beginning to be explored and practical implementation and testing has not yet been performed [47].

3.1 Emulation

Emulation [40] is a highly discussed form of preservation. An emulator is a software program that mimics the function of another computer [46]. Once a single platform has been emulated, it should be possible then to run and render any of the platforms software. The scalability of emulation has significant cost savings as many different types of formats can be run upon one platform.

Timing the development of the emulator is crucial. By waiting too long before developing an emulator, critical information required can be lost [46]. Conversely, developing an emulator too soon may result in the emulators becoming obsolete before they are of any use to us. Software Longevity techniques, as discussed further in this report, assist in prolonging the lifespan of an emulator.

A further problem of emulation is that by emulating the original platform, you are restricted to the technological constraints of that day. For example, the emulation of the BBC Domesday device resulted in a low resolution blocky screen, navigated only through the use of complex, non-intuitive techniques. Although emulating this device is a significant achievement and a demonstration that the principle works, the restrictions and limitations that such a method imposes questions its prolonged use. Is it worth preserving the functionality if it is a chore to use?

However, Holdsworth and Wheatley suggest without emulation it is unclear how we may preserve complex interactive digital objects [21].

3.2 Migration

Migration was originally regarded by some as the only feasible method of preservation; it is now considered by many as critically inadequate. Migration over extended time requires repeated transformations of the original bytestream when a format is in danger of becoming obsolete. These transformations are very slow and expensive, as every individual object must be transformed into the new format. Such transformations can introduce errors to the object, which may go largely undetected as it is infeasible to compare every individual file before and after each migration.

When transforming files, a vital technique to prove the file has not been altered is to reverse the migration. After a number of years and a chain of conversions, it may be beyond the resources of an institution to perform this check, so there is very little verification of the object that can occur. The resultant object can be significantly different from the original and, with no way to verify the integrity of the object, in many cases can be considered useless. The digital preservation community is beginning to understand that retaining the original bytestream is essential, allowing refinements to be made to a preservation method as more is learned.

3.3 Migration On Request

Migration on Request [34] manages to combine some benefits of both emulation and migration. Separate input and output modules allow similar formats to use a single tool that is maintained over time to preserve them. The object is archived in its original format and when required is input, processed and output in the designated format by the tool, as shown in figure 3.1. Discarding this output object after use and using the original bytesteam for later retrievals enable potential errors that would otherwise propagate through to be reduced.

A Migration on Request tool is written following a carefully engineered modular design, utilising software longevity techniques. The focus is on simplicity and understanding, so if a tool needs to be rewritten for a future platform it can be done so easily. An input module is used to convert the data object into an intermediate structure. This intermediate structure needs to encapsulate all the significant properties of the source and target formats. Once in this format,

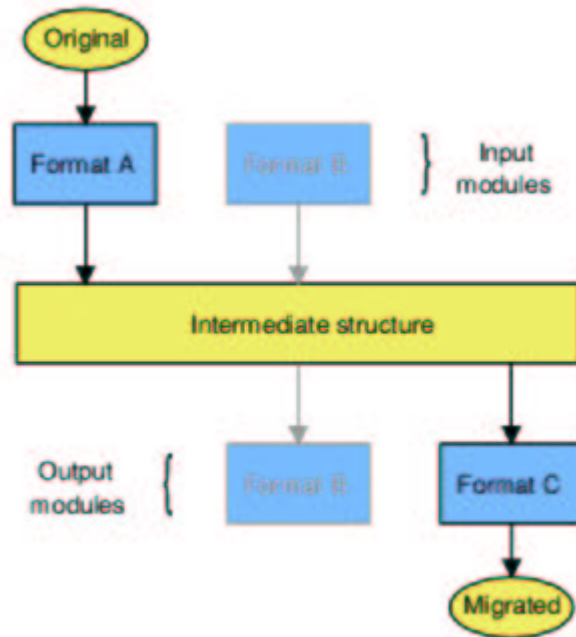


Image used courtesy of the CAMiLEON Project [9].

Figure 3.1: Over an extended duration, Migration on Request uses different modules

a corresponding output module will convert the intermediate format into the designated format for that technological day.

This means that an unsupported format only needs the input module to be written once, using a subset of the intermediate structures features. When a new output format is added, any new features of the format are added to the intermediate structures features. This may result in a large and complicated structure, but each input and output module would only utilise a subset of these. The difficulties arise when an obsolete input format has attributes that are not present in the designated output format. In this case, routines are needed to convert between these supported and unsupported features. There may be a loss of information in the chosen output, but this loss may disappear when a different output is chosen.

Although an output module could convert the unsupported features into something it could use (for example, Bezier curves into a series of small lines), this can result in unnecessary duplication of code. A more attractive solution is to implement these conversion routines in a separate, centralised process.

Thus, a tool consists of three modules: the input module, converting into an intermediate structure, the conversion module, ensuring the objects attributes map across from input to output, and the output module, correctly converting into

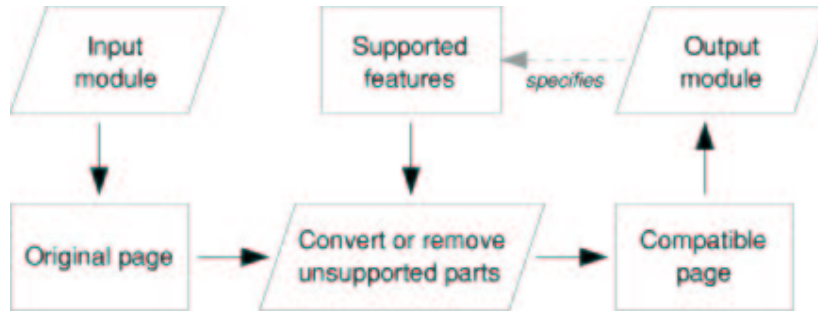


Image used courtesy of the CAMiLEON Project [9].

Figure 3.2: Stages of the Conversion Process

the output format. As shown in figure 3.2, the output module and the conversion module are closely linked: the output module will specify its supported features to the conversion module; the conversion module ensures an object conforms to these features. If not, the unsupported features are converted through the shortest chain of routines. To ensure infinite conversion loops are never created, it is only possible to convert to a “lower order” data element.

3.4 Longevity Of Solutions

Although there are currently reliable and efficient raster converters available these are written for today’s formats. Hardware and software are rapidly becoming obsolete, as NASA found with the data on their inaccessible magnetic tapes [45]. Therefore, I will be implementing this tool with an emphasis on longevity and simplicity. To do this, I will focus on conforming to well understood Software Engineering techniques.

Migration on Request inherently conforms to some such methodologies. Its modular design immediately makes the code divisible into sections and allows clear interfaces between components to be determined. The use of UML diagrams in the development phase will assist in determining these modules.

It is quite likely that a preservation method written today will need to be altered in some way if it is to be used in the future. We can limit the impact of this by carefully choosing the language in accordance to trends in usage. This is discussed in more detail in section 5.5. Clear comments should accompany the code in a conventional documentation style. Sun supply the document ‘Java Code Conventions’ [22] and while written for the Java platform can be applied to most languages.

Essentially, writing preservation tools in the early years of the fields developing

life is to assist future developers on their paths, so the use of well determined conventions is emphasised. Using these conventions will assist the removal of the complex and ambiguous nature of code. As most programmers have widely differing coding styles, the importance of this cannot be underestimated. Understanding another's code is a significant task; interpreting indecipherable code is much more difficult.

Libraries need to play an active role in the longevity of solutions. Extensive technical documentation needs to be retained to assist in future development. This will not enhance the longevity of preservation software, but rather enhance the longevity of the ability to produce preservation software. Archives, such as the Koninklijke Bibliotheek (KB) [28] or the Digitale Bewaring Project [12], are emerging to fulfil this role and the longevity of these institutions need to be ensured.

3.5 Summary

We have seen a number of proposed solutions to the problem of retrieving files. Some short term solutions, such as Printing to Paper or Migration can corrupt or remove the functionality of the file. However, these acts are being performed by institutions who need to take steps to preserve access to objects. It is essential to research and develop more appropriate methods such as Emulation and Migration on Request.

As Migration on Request is a new technique there are likely to be developments and refinements made to the method over time; producing more implementations will raise awareness of the areas demanding explanation and attention.

4. Authenticity

A prototype Migration on Request tool assists in planning for the accessibility of digital objects in the future. This is just one component of the set of problems faced in Digital Preservation. The document below details suggested solutions to another critical component: the authenticity of an object.

As the field of diplomatics shows, concerns about authenticity in sources are not new, however the “ubiquity of digital representations and the proliferation of source information [means] these issues are further complicated” [6]. Discussions regarding authenticity and integrity often result in controversial philosophical ideas linked to our conceptual understanding of documents in their legal, social and historical contexts. It must be distinguished what judgements can be made by code and what must be determined by humans.

It is worth defining two commonly confused terms within this field, authenticity and authentication. Authentication refers to establishing the identity of a user; that they are who they purport to be. Authenticity uses methods of identification and verification on a resource; addressing the integrity of a resource.

4.1 Trust

Perhaps the most important factor in determining the authenticity of an object is that of trust. For example, when we see exhibits in a museum we trust appropriate security has been employed by the museum and we are seeing the original.

It is very difficult to guarantee completely that the object’s integrity has not been compromised. Rather, a series of subjective evaluations are made based upon assertions regarding the object. In a digital archive, these assertions are contained within the provenance data of the object. Research is being conducted to establish formal methods that could be used to define rules about how trust and belief are assigned. This is a developing field. It will be interesting to see whether the cultural heritage community will establish a consensus on trust assignment rules [32].

Traditional methods of establishing the authenticity of an object require studying both the object and the collection of assertions associated with it. These assertions may be internal, such as a claim of authorship or date when published, or external, represented in (possibly third party) metadata accompanying the object. Several fundamental processes take place.

- The provenance of the object is examined and the extent to which we trust and believe this documentation is established.
- A forensic and diplomatic examination of the object is performed, to ensure its characteristics and content are consistent with the claims made about it.
- Signatures and seals that are attached to the object are investigated. These result in a high weighting of trust if they are consistent with claims and provenance.
- In some situations, we can compare the object to a trusted copy we have in existence. We assume this trusted copy has been through a similar verification procedure.

In the digital environment there are few forensics or diplomatics other than that of content itself. The distribution of copies to arbitrary parties and the abstraction of inscription (allowing a copy to be moved from hard disk to CD yet remain identical and authentic) causes change in the methods used to establish authenticity.

Perhaps the most fundamental difference is that it is not possible to preserve an electronic record, but only the ability to reproduce the record. There is inevitably a substantial difference between the digital representation of the record in storage and the form in which it is presented for use. Recall the object paradigm discussed earlier, where an object contains the data and the method to access it. The requirement of these software methods to translate the stored format into its rendered form entails the inevitable risk that regardless of how well the data was protected in storage the record may be inappropriately altered when the stored bits are retrieved and presented for use [23].

4.2 Archival Stages

It has become evident that there are several stages involved in establishing the authenticity of a digitally preserved object. Here, they have been reduced into three stages: The authenticity of the document before it is transferred to the custody of the preserver, the authenticity of the object whilst it is in the custody of the preserver, and the authenticity of copies produced by the preserver.

4.2.1 Before Ingest

Before objects are transferred to archival custody, the preserver must establish whether and to what extent the objects have been maintained by the creator

guaranteeing their authenticity, or minimising the risks the object could have undergone. The interPARES project suggest a set of benchmark requirements intended to support the presumption of authenticity [24]. When a preserver has established an acceptable level of trust in the object, it is common for the object to be archived along with several pieces of metadata detailing the basis of this trust. Often this includes a signature of the object.

Briefly, a signature binds a document to a person or authority. The signer uses their private key to sign an object. Their corresponding public key is stored in a Public Key Infrastructure (PKI) which is retrieved and used to decrypt the object. A noteworthy point here is that control of an identity can be transferred or shared with another simply by sharing the key pair. We have to trust not only the identity, but also their behaviour when in control of that identity.

In the physical world, a signature can suggest a number of possibilities. The signer may have:

- Authored the document.
- Witnessed the document and other signatures on it.
- Believed the document to be correct.

This shows the semantics of a signature can change. This is a difficult issue in the digital world, particularly as documents can undergo format changes. By reformatting a document, is the signature redundant? It appears we may need to include explicit statements detailing the meaning of each signature associated with an object. Clifford Lynch has detailed the need for such a vocabulary expressing the semantics of a signature [32]. In the same text he notes a potential future problem may be the occurrence of several public vocabularies, resulting in the complex problem of mapping and interpreting these.

The public keys need to be kept available for an indefinite length of time to facilitate access into the future. A trusted certificate authority allocates keys to people; Public Key Infrastructures (PKIs) store them. In the long term scenario, some very important problems have appeared. Most key pairs have expiry dates, and PKIs could potentially disappear. Methods need to be formalised to determine if a key should be reissued, and long lasting well documented infrastructures need to be set up.

Asserting the authenticity of an object gives us proof the object is what we believe it to be. It is also important to ensure the object has not been corrupted over time. We call this the integrity of the object; to check this we often use the digest of an object. This value is stored along with the object; on retrieval a new digest is computed from the document in hand. The original and new digests are compared and if they are identical the object has not been corrupted. The digest algorithm is carefully designed to make slightly different pieces of data

have widely different digests. The original document cannot be recreated from the digest. However, this returns us to the original problem: Our confidence in the object can only be as good as our confidence that the digest has not been manipulated.

Conversely we can look on the digest as a very fragile mechanism. If our newly computed digest fails, all we now know is that the original and current objects are not identical. It would be more useful to know the object we have retrieved from an archive is nearly the same, or whether it has been significantly altered. Separating an insignificant alteration from a significant one may well be impossible to ascertain; developing a deterministic method to establish this would be even harder.

Thus on ingest to an archive, the object would have associated to it the digest of the object, the algorithm used to compute the digest, a timestamp and the certificate of the author. These would all be signed by the author's private key.

4.2.2 During Archival Custody

Once the object is in archival custody the preserver needs to maintain its authenticity. This requires installing security programs preventing unauthorised users from altering the object and the archivist to follow guidelines when treating the object. The ideal situation for an object in storage is to store the object in a secure place where it can be retrieved and accessed without the object ever having been altered. Accidental alterations can arise when refreshing the medium the data is stored on. If a document is legally admissible, the threat of alteration from corrupt users is greatest while the document is in the archive. Append-only logs assist in preventing unauthorised attacks on documents; many other management and security techniques are also needed.

A users trust in the archiving institute will help establish their trust in the integrity of the object. Provenance metadata on the object will assist our assurance that the object's integrity has not been compromised, however we only believe this metadata if we trust the supplier of it. Faith in archiving institutions is generally built around respect for the methods used by them and often historical evidence detailing the institutes honesty in the past.

4.2.3 Upon Retrieval

The problem arises when, as we are discussing here, the object is being transformed upon retrieval to access it at that time. The user is then left with the question "how do I know this is what it purports to be?" Fundamentally, this

again comes down to trust in the institution (and thus the software) acting on the object.

Canonicalization [31] uses computational algorithms that are used to extract the fundamental meaning of objects according to definitions of what this meaning constitutes. This meaning is determined by the significant properties of the object. This method assumes we are concerned with the integrity and authenticity of the meaning of the object, rather than the literal bits themselves.

A Migration on Request tool is a practical example of such an algorithm. We are attempting to extract the meaning of the object and transport it to a new format. With images, the significant properties seem quite simple to define: the end visual product; textual documents contain format and typeset issues that are not so simple. The difficulty is precisely defining and agreeing upon both the properties and the method to extract it from an object.

Although we can agree on these properties transformations are still dangerous. A transformation can implicitly impart a particular point of view and result in certain characteristics being emphasised or obscured. Slight modifications in numbers (such as the rounding of decimal places) can cascade and result in a vastly different object.

If transformations are the only way to access an object as we assume they will be, we need to determine their authenticity now the object has been changed. Transforming through the canonical form implies that the meaning has also been transported. However, the original signature should not apply to the reformatted object, as the original signer may not be present to witness it. Perhaps a more appropriate method is to ask the original signer to sign a canonicalization function C acting on the object D : $C(D)$. We can then reformat the object as long as the changes remain within the bounds of the canonical form. Verification digests can be computed over $C(D)$ rather than D . We can also verify a reformatting program R by comparing $\text{HASH}(C(D))$ to $\text{HASH}(C(R(D)))$ and ensuring they are equal. An appropriate canonical form will allow comparisons of the objects integrity without requiring the specific ordering and organisation of their internal structure is maintained.

Clifford Lynch explores this topic in greater detail in his paper Canonicalization: A Fundamental Tool to Facilitate Preservation and Management of Digital Information [31]. Using such methods theoretically allow us periodically to implement and review an output format and ensure that the data is still accessible and the integrity has remained sound.

4.3 Summary

Throughout this chapter we have seen the variety of authenticity issues faced in the preservation of data. Once the integrity has been assured before ingest to an archive, care must be taken to maintain it. Retrieving it is naturally the biggest challenge. Migration on Request follows the principles of canonicalization assisting us to maintain integrity and allows us to develop the tool over time.

5. Design

5.1 Objectives

This project is intended to investigate the difficulties faced when implementing a Migration on Request tool. I hope to highlight the advantages of this method related to the associated digital preservation issues; Alternately I would regard this project a success if I were to highlight any disadvantages when using this technique. Primarily, I am interested in raising awareness in both good and bad practises for future implementations of such a tool. To do this I intend to develop the tool using Software Engineering techniques ensuring a modular design.

Writing a Migration on Request tool is still a new task. Converting between unsupported elements is not a task to be undertaken lightly and conceptually can be difficult to think about in a clear manner. Therefore, it is important to choose the file types and formats for inclusion in a prototype tool carefully.

The category of files is the first stage in determining the tool. For a prototype with such a limited timescale the size of the project must be restricted. Therefore, some file types can be ruled out. Audio and visual files are out of the scope of this project. The suggestion remains that emulation may be the only way to secure the long term access of complex interactive objects [21]. Word processed documents would be a very interesting area to tackle and feasible as long as the conversion of only a limited subset of functional abilities were attempted. Covering complex features such as style sheets would be too significant for a project of this scale. If we cannot view the proprietary specifications, which are currently unavailable, it may be impossible.

Attempting a graphics format has many advantages. The visual output allows an immediate evaluation of the conversion. Many graphics formats are similar in structure, enabling the development of a chain of unsupported element conversions. Graphics conversion is a popular field; this means mailing lists can provide me with an extra level of support from the developers of such programs.

Raster graphics formats appear deceptively simple. Fundamentally, they are simply an array containing intensities that pixels must be lit to. However, compression techniques and varying colour depths mean they vary quite considerably. The complexity of such formats is also at a level I feel comfortable with. Therefore, the practical element of this project will be to develop a raster graphic Migration on Request tool.

It is important to reiterate the difference between the Migration on Request

tool to be implemented and the numerous graphics converters available on the market. Graphics converters presently available are aimed for use today. As has been explained throughout this document so far, any implementation attempted is to be used in the long term. The Migration on Request structure has canonical properties geared towards making integrity verification easier. Although ensuring these characteristics occur can initially look like a simple task, careful design and implementation work must first take place.

5.2 Big and Little Endian

Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory [43]. Big-endian is an order in which the “big end” (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the “little end” (least significant value in the sequence) is stored first. For example, in a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage (if 4F is stored at storage address 1000, for example, 52 will be at address 1001). In a little-endian system, it would be stored as 524F (52 at address 1000, 4F at 1001).

Early Macintosh processors, made by Motorola, use the big-endian method for their internal representation. Intel and AMD processors use the little-endian method. PowerPC’s can use either, although the operating system imposes one convention or the other. For example, current Mac PowerPC’s use the big-endian representation.

A simple analogy is that internet domain name addresses and e-mail addresses are little-endian. A big-endian version of a domain name address would be `uk.ac.ed.www`.

File formats can store numbers in both big- and little-endian format. While it is quite simple to convert between the two types, care has to be taken to ensure we are working with the correct order.

5.3 Raster Formats

There are a number of differences and developments that have occurred in raster graphics formats.

Colour Depth This determines the number of colours available to the image. GIF images are restricted to a palette of 256 colours, whereas newer formats

such as PNG can handle up to 48 bits of information- to our eyes they are virtually unlimited. Development in fields such as medical imaging mean this will soon be surpassed.

Colour Palette A palette is a set of colours that can be used within the image. These can contain as few as two colours and can be fixed or allocated. It is essential to ensure that colours available in the original image are also available in the converted image.

Transparency Has a varying level of support among formats. JPEG does not support transparency, GIF has a binary range which has been known to cause display problems. Formats such as PNG and TIFF provide full support, with a sliding scale of transparency allowing opaque images.

Compression There are a number of compression techniques used in raster graphic formats; these can be lossy and non-lossy. Lossy techniques, such as the JPEG algorithm eradicate information with little deterioration when viewed. However, repeated applications of the technique can result in a drastic reduction in quality. Non-lossy techniques, such as those used in GIF and PNG look for recurring patterns in the image data and abbreviate them. The non-lossy LZW compression algorithm used in GIF images is patented by Unisys [44]. The legal ramifications restrict the use of LZW compression as a long-term preservation technique. The compression technique used in PNG achieves similar compression ratios, yet uses an established open standard method.

It is important to choose the formats covered in this implementation carefully. As seen earlier, there are two module variations that must be catered for: input and output.

5.3.1 Input Formats

The past decades have seen a number of developments in raster graphics formats. A significant addition is the proliferation of compression techniques; cyclic images such as those found in GIF and transparency are also noteworthy recent developments.

To test a Migration on Request tool, it is advisable to write input modules spanning the lifespan and range of features found in the file type. The following sections discuss potential input formats.

5.3.1.1 The PNM family

The PNM family of files was originally developed as a set of lowest common denominator file formats [26]. There are two variants of the family, an ASCII text format and a binary equivalent. The family consists of three different format types; the Portable BitMap (PBM), the Portable GreyMap (PGM) and the Portable PixMap (PPM). Note that text files are neither big- or little- endian.

Each format follows the same basic layout. Whitespace separates each element of the file. A two character “magic number” indicates the format type, this ranges from P1 to P6. The width and height (formatted in ASCII decimal) are the next data values encountered. Comments begin with a ‘#’; as there is no delimiter the rest of the line is classed a comment. PGM and PPM files specify the colour depth of the image, again formatted in ASCII decimal. PBM files are bitonal, so there is no need to specify a depth.

The above information comprises the header, and is the same for both the binary and text formats. Following this is the image data itself. In the ASCII format, each element is separated by whitespace. The image starts from the top left, and progresses row by row, left to right, moving down the image. PPM files specify each value as a RGB triple, PBM and PGM only have one value per pixel.

5.3.1.2 GIF

The CompuServe Graphics Interchange Format (GIF) [10] has become a very popular network graphic format, primarily due to the non-degrading high compression algorithm. The format is, however, very limited. It supports a maximum of 256 colours which are determined by a palette. The Lempel-Ziv-Welch (LZW) compression algorithm is patented by Unisys who have begun demanding royalties and licence fees from successful applications [44]. The limited, cartoon like colours, small size and ability to cycle through sets of images makes it suitable for web advertisements.

Two versions of the GIF format are in use, versions 87a and 89a. 89a is more common and is backwards compatible with the earlier version. The header allows us to determine which we are using and skip over any incompatible blocks where necessary. A GIF file is stored in little endian format, and broken up into blocks with each block having a maximum size of 256 bytes. The compression licensing issues have steered me away from further development for a prototype tool, however it is worth noting the cyclic image feature which must be incorporated into the design.

5.3.1.3 QuickTake

QuickTake was a format used by the Apple QuickTake 150, one of the first budget digital cameras [3]. It is essentially a PICT image file, and similarly uses big-endian order, but requires a special codec to view the image. The codec was not distributed in Mac OS X, so users must convert a QuickTake file in OS 9 to TIFF or JPEG [4]; however, even in OS 9 the codec is sometimes not present.

As yet, it has not been possible to obtain the specification for the QuickTake format or the codec. Developing an input module for a Migration on Request tool would be invaluable to the preservation of stored images used in this format, as soon we may no longer have access to OS 9 to perform transformations.

5.3.1.4 JPEG

JPEG is an acronym for the Joint Photographic Experts Group. Often when we refer to JPEG image files we are referring to the JPEG File Interchange Format [17]. A JPEG data stream is a byte-oriented data-stream compressed using a lossy technique which stores data in the big-endian storage format. Lossy compression means a varying amount of the image information is thrown away, never to be recovered. Because of this, the compression algorithm can attain high compression ratios with only a small loss of resolution. However, repeatedly storing a JPEG image, opening and then storing it again causes serious degradation of the image. The lossy compression techniques make JPEG's unsuitable for use as an output format.

JPEG uses a YC_bC_r , or luminance/chrominance colour space representation. This is a function of brightness (Y), blueness (C_b) and redness (C_r). It works on the principle that the human eye is more sensitive to changes in brightness than it is to changes in colour.

To retrieve an image, the following steps are needed [11].

- Decompress the Huffman or arithmetically encoded data.
- Unquantize the DCT data, inserting zeros for truncated values.
- Apply a Reverse Cosine Transform on the DCT data.
- Reverse the subsampling of the pixel data.
- Convert from luminance/chrominance colour space to RGB colour space.

5.3.2 Output Formats

The output formats that are written today are used to verify the tool is not corrupting or altering the significant properties as the data is processed. We cannot expect any of the formats we use today to continue to be in use fifty years in the future. This means the output modules I write will be thrown away; new ones that are written in the future may also be thrown away and so on. In practice archive managers have to expect to write an output module at the present time that output module is needed.

When trying to determine what the best output format would be, there are a number of factors to be taken into consideration. A completely open source specification is essential, as it ensures developers both now and in the future achieve a thorough understanding of the formats. The formats should also be chosen according to their expected longevity- it would be nice to reduce the need for more output formats to be written if a few well chosen modules would suffice.

A potential format for future development may be JPEG 2000. This format has not yet been finalised, but looks to become a well supported successful format [27].

5.3.2.1 The PNM family

To ensure the intermediate structure is not altering the file as it passes through the intermediate structure, I plan to implement an output module for the PNM family. This should be very simple to achieve and works as a good test of the intermediate structures integrity.

5.3.2.2 PNG

The Portable Network Graphic (PNG) format was developed to replace the CompuServe GIF format. The developers sought to create a format that could handle a wide range of graphics that current technology supported while leaving room for next generation developments [11]. It is an open standard; the compression algorithm, a version of Deflate found in PKZIP, is publicly available. This makes PNG free of royalty and licence fees. The supported colour depths are also impressive: PNG supports colour palette-based images, 16-bit greyscale and 24-bit/48-bit true colour images. When using the 8 or 16 bits of alpha channel also available, PNG can handle 32-bit/64-bit true colour images. PNG stores data using big-endian byte ordering.

The PNG file structure begins with an eight-byte signature, uniquely identifying the file as a PNG file. Following this is a series of blocks, called chunks. Each

chunk has the same format, and the maximum size of a chunk is restricted to $2^{31}-1$ bytes. Three chunks are required in every PNG file: the IHDR, header chunk; the IDAT, image data chunk, of which there may be more than one in the file placed in sequence; and the IEND, image trailer block. The PLTE, colour palette chunk should be present if the image is palette based.

A PNG reader must be able to decode these three chunks. Others are optional and a reader can usually safely ignore the blocks and be able to create a recognisable image. The PNG specification [14] further details the available chunks that may be encountered and/or used.

All integers requiring more than one byte must be in network byte order: the most significant byte comes first, then the less significant bytes in descending order of significance (MSB LSB for two-byte integers, B3 B2 B1 B0 for four-byte integers). The highest bit (value 128) of a byte is numbered bit 7; the lowest bit (value 1) is numbered bit 0. Values are unsigned unless otherwise noted. Values explicitly noted as signed are represented in two's complement notation.

Each chunk consists of four parts:

Length A 4-byte unsigned integer giving the number of bytes in the chunk's data field.

Chunk Type A 4-byte chunk type code.

Chunk Data The data bytes appropriate to the chunk type, if any. This field can be of zero length.

CRC A 4-byte CRC (Cyclic Redundancy Check) calculated on the preceding bytes in the chunk, including the chunk type code and chunk data fields, but not including the length field. The CRC is always present, even for chunks containing no data.

5.3.2.3 TIFF

The Tagged Image File Format (TIFF) [2] has been suggested as a valuable long term storage format, due to its flexibility, open nature and ability to store images uncompressed. It can store data in both the big- and little-endian data formats, and has a flag denoting which. A TIFF is composed of a series of Image File Directories (IFD), allowing multiple images to be stored within one file. Each IFD is composed of an array of Word-sized (4 byte) tags and ends with a pointer to the following IFD in the TIFF file.

Most TIFF images are compressed using one of the extended TIFF algorithms such as LZW or JPEG; the ability to use alternative compression techniques plays a significant role in the appeal of TIFF.

A simple baseline TIFF file containing a single image would consist of the TIFF header and one Image File Directory. The IFD would contain the following tag fields: Bits/Sample, Colour Map, Compression, Image Width, image Length, Photometric Interpretation, Resolution Units, Rows/Strip, Strip Offsets, Strip Byte Count, X Resolution and Y Resolution.

5.4 Significant Properties of Raster Formats

Many file types have complex underlying features that vary from format to format. Raster formats have such features, for example the varying compression techniques that can be used. However, I would not classify compression techniques as part of the objects significant properties and so these should not be stored within the intermediate structure.

Below are the significant properties I have determined and aim to preserve.

- The dimensions of the image. This includes the width, height and resolution.
- The colour depth of the image.
- If the image is a paletted image, the specific colours of this palette.
- Transparency must be preserved if supported.
- The image data itself must also be preserved and unaltered.

With raster images, viewing the image allows the majority of the significant properties of the image to be determined. However, this is only from my personal perspective. Other institutions may have other properties that they would require to be retained. For example, I do not see the need to retain the compression technique used to store the data; others may.

I also do not think it necessary to reinsert comments in the output image. However, upon retrieval from an archive such comments might give valuable diplomatic and provenance data that had not otherwise been recorded. I plan to allow the easy extraction of such comments despite their exclusion from the final output image.

5.5 Languages

The programming language used for a long-term Migration on Request tool should be chosen carefully. Primarily it should be chosen for longevity; it should

also be chosen to logistically apply to its intended use and to ease future understanding and interpretation.

Attempting to determine the longevity of a language can only be a heuristic evaluation. It can be assisted by the quantity of software written for and used by consumers. The use of new, unestablished languages is inadvisable: without a large support languages can quickly die out, requiring the tool to be rewritten for newer languages. An example of such a language is Python. Although gaining popularity amongst programmers and older than Java, it does not have the support and stability to be used in this context.

5.5.1 C and C++

C is arguably the most successful programming language ever. It was released in the early 70's and remains one of the most popular languages in use today. The success of Unix was an important factor in its ubiquity, as was the portability of the language. One of the most attractive elements of C is that it doesn't try to achieve too much: it covers the needs of its programmers, yet remains simple and small enough to remain elegant.

C++ was developed during the 1980's when object-oriented programming arrived. As it is a superset of C it shares many of the same features. However, a common criticism is that C++ is C with an underdeveloped object-oriented nature tacked on.

The quantity of applications written in C imply the language is here to stay for quite a while yet, which is appealing for its use in long term tools. To extend the longevity of the language, David Holdsworth suggested a subset a C, known as C--, for use [19]. Research conducted by Phil Mellor suggests that C-- may be more suitable for writing Emulators rather the Migration tools [33]. C has a history as a system implementation language, but for the rich data structures found in file formats there are more attractive language solutions.

5.5.2 Java

Java, released in 1994, has achieved tremendous popularity since its release. This has occurred in part due to its ease of learning, causing many universities to replace C as the language taught, and due to the proliferation of networks across which Java works particularly well.

Java operates entirely within the object-oriented paradigm. This means that each data structure, known as a class, has a set of methods that can act on the data encapsulated with it. In a Migration on Request tool, we are breaking

a data file into a hierarchy of related components; conceptually this also fits the object-oriented paradigm. Using Java to write a Migration on Request tool would be a valuable experiment. The only prior implementation of a tool was written in C [34]; the project was a success, but the conclusion was that it may have been simpler and less cumbersome to write it in Java. Where longevity and extendibility are such important factors, the tool must be as lightweight as possible.

5.5.2.1 Java--

As Java is a new, rich language it is expected to undergo several revisions during its lifespan. This may result in the removal of some classes from the language. To combat this I intend to use a subset of Java called Java--. The large number of classes included in Java means it is not possible to exclude certain aspects of the language as it was for C--, rather I intend to use a minimal set of classes, recording and justifying inclusion where I require more classes. This documentation will allow further developers to find and replace any deprecated classes that may occur, while keeping the potential number of such classes to a minimum.

5.5.2.2 Garbage Collection

When data is no longer required in C, it must be explicitly deleted by the programmer. Java has automatic garbage collection, so no explicit statements are included denoting where data is no longer needed.

Instead objects created within methods are automatically marked as unusable when the method has finished. The objects are removed when the garbage collector is next run. If we assume automatic garbage collection will continue in languages, this is not a problem. Rewriting a C program requires the removal of such delete statements (and also has the problem of translating to an object-oriented paradigm). If garbage collection does not continue rewriting a Java program would require inspection to determine where data is no longer needed.

5.5.2.3 Personal Considerations

Edinburgh University uses Java as a foundation for its degree course, consequently I feel more comfortable writing in this language over C. I feel it would be an interesting experiment, and one within my bounds, to attempt this prototype using Java.

5.6 Development Considerations

Chapter 3 discussed Migration on Request as a preservation method; here we detail the specific design constraints when developing such a tool. A considerable benefit of the method is when a new output format module is written, that output format becomes available for every file in the archive. To ease future development of output modules, the intermediate structure should be carefully and intuitively designed. Unfortunately, the intermediate structure may grow to be large and cumbersome as it must encompass all the elements of the input formats, which change as time progresses.

It is often possible to represent elements in different ways; the proprietary and secretive nature of many formats causes differences to be common. Often representation methods are interchangeable- as long as a representation method does not lose information it does not matter which we use. Phil Mellor proposed the following techniques to determine the inclusion of representation methods.

Suppose File A represents an object with Method A and File B represents an object with Method B. If conversion between Method A and Method B can occur without any loss of information, we can store in the intermediate format with Method A, Method B or even Method C, assuming Method C also has no loss.

If File A can represent an object with either Method A or Method B, then we must include methods for both in the intermediate structure, along with a flag denoting which was used. Similarly, if we cannot convert between the methods of File A and File B, then we must store both methods.

Care must be taken to ensure we don't retain redundant information. This can clutter the intermediate format, increasing its complexity, the chance of bugs and inconsistencies while reducing the intuitive nature we sought.

5.7 Intermediate Structure

The design of the intermediate structure, as shown in figure 5.1, is intended to be as generic and extendible as possible. A **Document** object encapsulates a **Page** object, and records information such as the author, the creating program and any comments it may come across. Currently, this metadata is simply discarded when the object is deleted, but it would be simple to output to a text file to assist provenance data.

Most images only require a single **Page** object, but allowing multiple **Page** objects caters for multiple images contained within a single file. A **Page** object is a

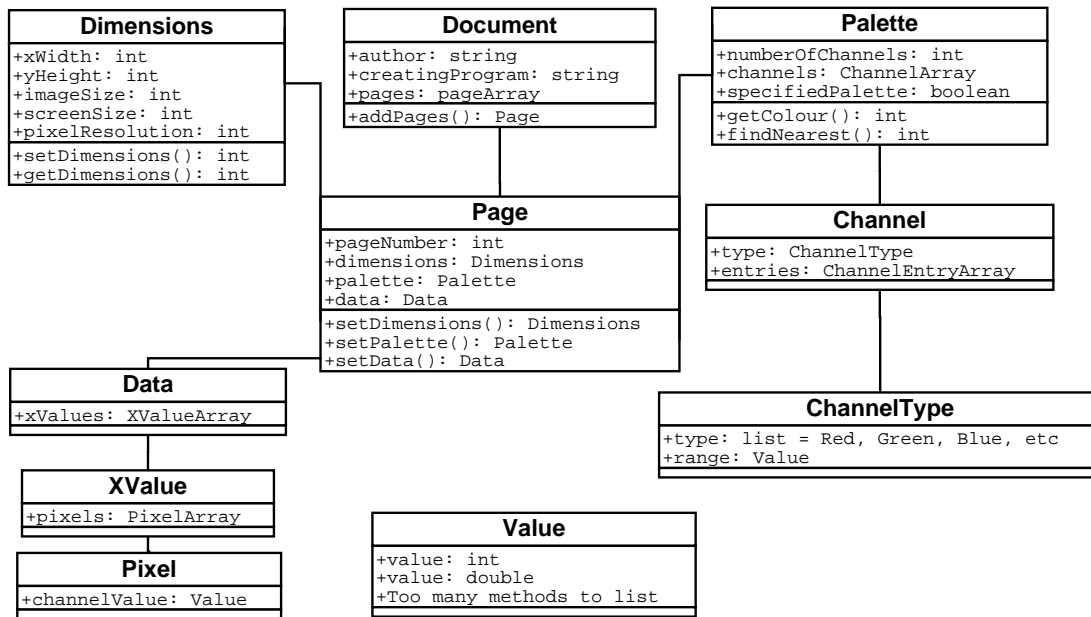


Figure 5.1: Detail of the Intermediate Structure

wrapper and contains the `Palette`, `Dimensions` and `Data` objects along with a record of the `Page` number.

The `Dimensions` object contains the significant properties related to the image size. The fields of the `Dimensions` object are `xWidth`, `yHeight`, `imageSize`, `screenSize`, and `pixelResolution`. The pixel resolution information determines the size of the image, so must be retained. From this information, we should be able to replicate the image size and resolution.

The `Palette` object contains an array of `Channels`. A `Channel` object records a palette when necessary, and includes a `ChannelType` object providing more detail. To record the palette, an array of entries is created and the palette values inserted in here. The `ChannelType` object records the type of the channel, this method allows RGB, CMYK and Transparent channels and can be extended where necessary. Absolute colour spaces do not need an array of entries in the `Channel` object, it is enough to record we are using absolute colour spaces.

Figure 5.2 shows the `Data` structure in detail. This structure contains the actual image data. The `Data` object contains an array of `XValue` objects, determined by the height of the image. The `XValue` objects contains an array of `Pixel` objects, determined by the width of the image. The `Pixel` object contains an array of `Values`, determined by the number of channels the image uses. These `Values` then take the colour of the pixel. It has been developed this way to allow maximum flexibility, as some image types start the image from the top left point and others from the lower left. This structure allows flexibility over how values are inserted

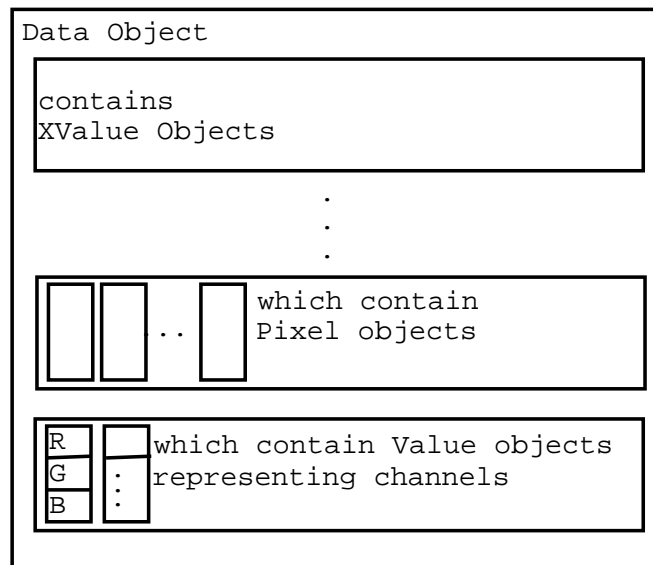


Figure 5.2: Detail of the Data Object

and retrieved from the image.

5.8 Unsupported Attributes

Many formats share similar features, however as time has progressed so has the technology. Many early formats are very basic when compared to modern formats. The number and variety of features included in current formats can cause problems when mapping from one format to another.

When parsing a file, if an attribute of the intermediate format does not have a corresponding input value it should remain undefined rather than allocated an arbitrary value. The output format may not require this attribute either; designating arbitrary values can alter the perspective the document is viewed from. Only when the output format requires a value should the program attempt to fill in the blanks.

When a Document object contains features the output format cannot handle, there are several options we have to deal with this.

- Ignore unsupported features. This is not a valid solution to the problem. Features would be excluded from the output file and the integrity of the file would be compromised.
- Each output module converts unsupported features into something it can interpret. Having the conversion algorithms in each output module causes

Type	Contains	Default	Size	Range
<code>boolean</code>	<code>true</code> or <code>false</code>	<code>false</code>	1 bit	NA
<code>char</code>	Unicode character	<code>\u0000</code>	16 bits	<code>\u0000</code> to <code>\uFFFF</code>
<code>byte</code>	Signed integer	0	8 bits	-128 to 127
<code>short</code>	Signed integer	0	16 bits	-32768 to 32767
<code>int</code>	Signed integer	0	32 bits	-2147483648 to 2147483647
<code>long</code>	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
<code>float</code>	IEEE 754 floating point	0.0	32 bits	$\pm 1.4\text{E-}45$ to $\pm 3.4028235\text{E+}38$
<code>double</code>	IEEE 754 floating point	0.0	64 bits	$\pm 4.9\text{E-}324$ to $1.7976931348623157\text{E+}308$

Table 5.1: Java Primitive Data Types

duplication of code, although conversion into the specific attributes of the format should be done here.

- Develop a centralised set of routines to convert between intermediate format types. This should be contained in a structure along with details of the features supported and required by a module. A `Document` object can be passed to this structure where unsupported elements are downgraded into a desirable form; the structure also ensures all the required output features are present, assigning pre-designated values where necessary.

The returned `Document` is now a complete, well-supported object ready to be output. Here the `Document` can be passed to the second option on the list, where conversion into the specific attributes can occur.

Within the conversion structure it is not necessary to support conversion between all similar attributes. As long as information is not lost, a hierarchical chain of conversions will suffice. This reduces duplication of code and eases development when new boundaries are established. For example, a fully connected set of n conversion routines would require n new methods to support a new output feature. However, a hierarchical structure simply requires 1 new method.

5.9 Number Systems

Java supports eight basic data types known as primitive types, as shown in table 5.1. Of particular interest is the `byte`, `int` and `double` types. Data is commonly stored as a stream of bytes, however their small range often make these restrictive to operate with. Once the image data has been read in, it will be stored in the

appropriate form, most commonly either an `int` or a `double`. The only difference between the integer types `byte`, `short`, `int` and `long` is the number of bits and, therefore, the range of numbers that each can represent.

Real numbers are represented with the `float` and `double` data types. A `float` is a 32-bit single precision floating point value, a `double` is a 64-bit double precision value. A floating-point value can be represented by a string of digits followed by a decimal point and then another string of digits (e.g. 123.45 or 0.0). A floating-point literal is a `double` by default, and uses exponential notation where a number is followed by the letter `e` (or `E`) and another number. The second number represents the power of ten by which the first number is multiplied (e.g. $1.2345E02 = 1.2345 * 10^2$).

Java allows conversions between integer values and floating point values. There are two basic types of conversion. A widening conversion occurs when a value is converted into a type that is represented with more bits, and therefore has a wider range of legal values. Narrowing conversions are not always so safe, and occur when a value is converted to a type with fewer bits. For example, it is not reasonable to convert the integer value 13000 to a `byte`, as `byte` can only hold numbers between -128 and 127. Although the Java compiler will not allow narrowing conversions, you can force the conversion by performing a cast. It is important to look at the number systems of each format; for example, 32-bit integers must be converted into a set of 4 8-bit bytes before written to a PNG file.

5.10 Summary

The primary objective of this project is to raise awareness of both good and bad practices when using Migration on Request. An overview of raster graphics was presented, followed by a detailed look at some graphics formats. While some of the formats discussed will not be included in the Migration on Request tool, most of the features that need to be included in the design are encompassed by these formats. Having used these descriptions to establish the significant properties of raster graphics formats, a detailed and complete design of the intermediate structure was presented.

6. Implementation

6.1 Implementation of Structure

The previous chapter explained the structure and form the tool was to take. Translating this into a real world system required some carefully made decisions.

I decided to use a command line interface. It seemed unnecessary to work on creating a graphical user interface. I would predict the graphical elements of Java are among the most likely to change over time as developments are made.

A main class was required, this ensured all arguments were valid when using the tool and provided some command line help. When the program was run, this class called the correct input module which returned a **Document** structure. The **Document** is passed to the **Support** structure, where its validity for the chosen output format is automatically checked. Any necessary conversions are performed and required values are filled in. The validated **Document** is returned to the main class, where it is passed to the relevant output module.

The main difference between figure 5.1 and the real world implementation lies with the **Document** object, although even this is quite conceptual. Rather than being a static structure that is filled in, the **Document** is passed around the modules and dynamically altered accordingly.

6.2 The Value Class

With the focus on longevity, I tried to make the code as modular as possible. When dealing with values, such as the colour depth and dimensions, I decided to wrap the values in a **Value** class. This simple class wraps **doubles** and **ints**, and allows simple operations to be performed on them. Having this **Value** class allows modifications to be made to the number systems of the language, theoretically only requiring modifications to the **Value** class for the entire program to run correctly.

The drawback lies with the decrease in efficiency and the increase in size. For a reasonable sized image hundreds of values will be read in, each requiring a **new Value** to be created. Any arithmetic operations that are required are slower than the inbuilt operations of the language; however as the focus of the project is on integrity and longevity the loss of efficiency is a justifiable sacrifice.

The `getType()` method provides a distinction between the different data types that are encapsulated, so data can be manipulated accordingly. Different data types can be added, allowing easy extensions to the class. Note that adding new data types will require some alterations to the `Value` method definitions.

6.3 Conversion between Unsupported Elements

I determined the following as a set of graphics attributes where conversion may be a difficulty. I have described the solutions I used, however there may be alternatives which arise over time. Whenever we perform a conversion we need to assume an intelligent user who is aware of the need to prevent a loss of information. For example, we assume a user would not convert a 24-bit JPEG file, with 16 million colours, into a GIF file supporting 256 colours. However, sometimes the user has good reason to drop information. In the case where a loss occurs, the user should be informed of the risks and asked whether they wish to continue.

6.3.1 Continuous to Paletted

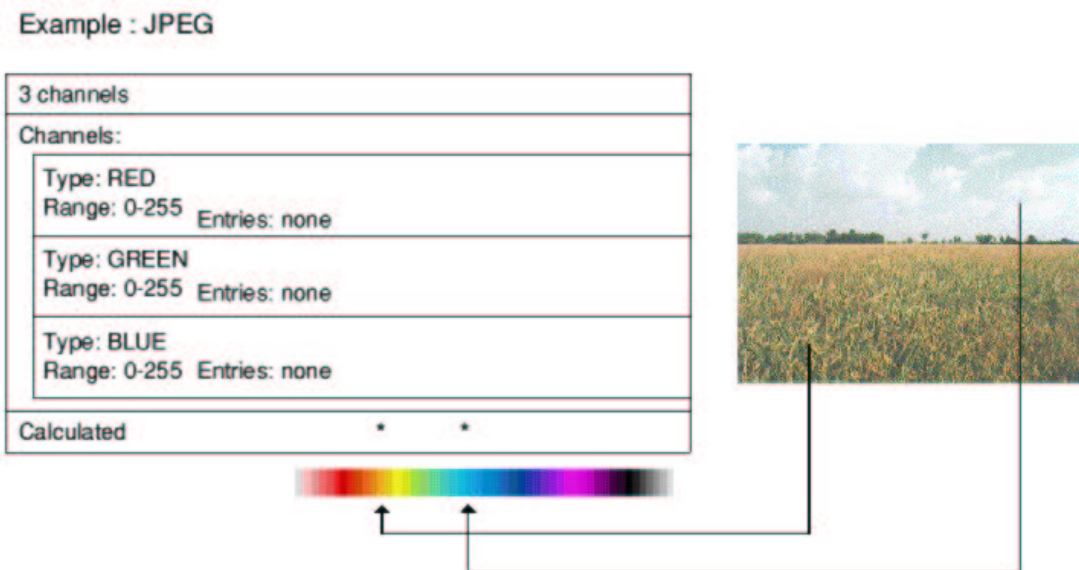


Image used courtesy of the CAMiLEON Project [9].

Figure 6.1: Continuous JPEG as stored in the Palette object

The `Palette` object design allows easy transformation from continuous to a paletted representation. Transformation from a palette to a continuous format is not allowed. It would be very difficult, if not impossible, to reproduce identical colours

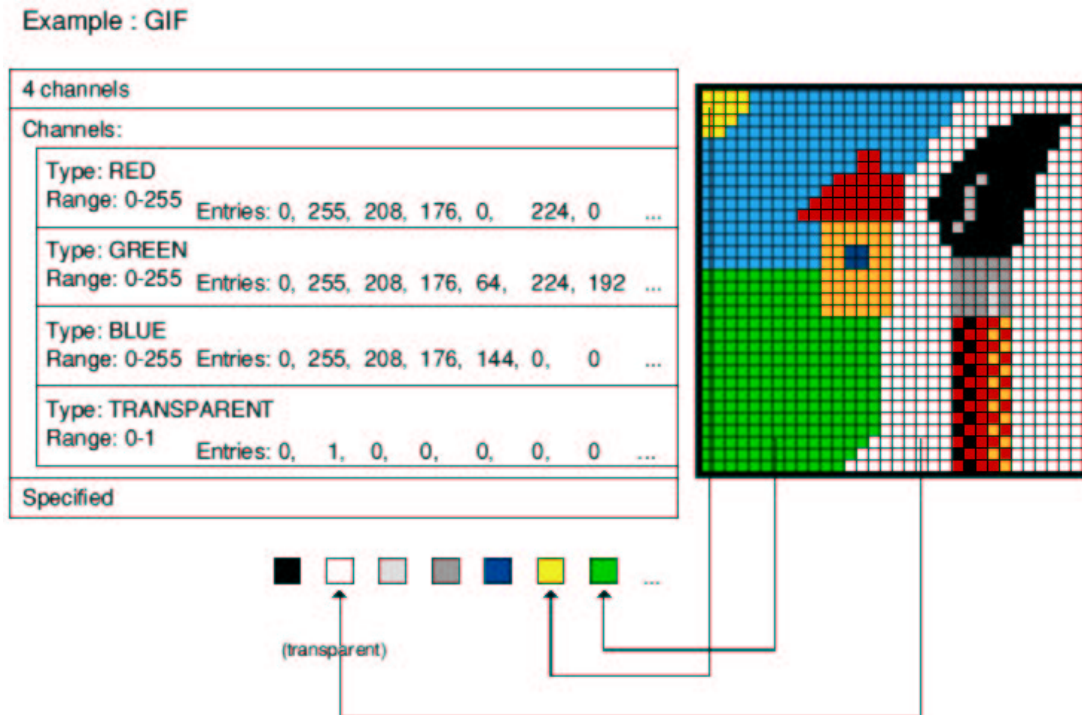


Image used courtesy of the CAMiLEON Project [9].

Figure 6.2: Paletted GIF as stored in the Palette object

for the output format and would generate a loss of information. A continuous format does not require the entire palette to be stored. As we are dealing with one channel at a time, we can simply record the range of each channel (see figure 6.1). Storing the palette when required is also intuitive (see figure 6.2).

Constants have been defined, such as `ChannelType.GREYSCALE`, `ChannelType.RED` etc. These allow easy differentiation between channels. Note that more constants can easily be defined to extend the model. When handling a continuous image, the intensity will equate to the value within the range. E.g. for a range of 0-255, the minimum intensity equates to 0 and the maximum intensity to 255. As the values are continuous (like a sliding scale), it is not necessary to store the actual values.

6.3.2 Colour Depth

For this prototype, it is necessary to assume a relatively intelligent user who will not be converting into a format with a lesser colour depth (eg. 24-bit to 16-bit). When dropping into a lower colour depth, there is little the developer can do to prevent loss. A careful selection of output formats will prevent the ability

to drop colours and may be the best solution. For example, PNG and TIFF currently have high-end colour depths. Technological advances in colour depths would require the replacement of these output formats.

6.3.3 Bitonal/Greyscale/RGB/CMYK

A chain of upwards conversions can take place between Bitonal, Greyscale and RGB, but converting back down will lose information. RGB can convert back and forth YC_bC_r without loss. Conversions between CMYK and RGB can result in the loss of colour clarity.

The following equations show how to convert between RGB and YC_bC_r .

$$Y = 0.299R + 0.587G + 0.114B \quad (6.1)$$

$$C_b = -0.1687 - 0.3313G + 0.5B + 128 \quad (6.2)$$

$$C_r = 0.5R - 0.4187G - 0.0813B + 128 \quad (6.3)$$

$$R = Y + 1.402(C_r - 128) \quad (6.4)$$

$$G = Y - 0.34414(C_b - 128) - 0.71414(C_r - 128) \quad (6.5)$$

$$B = Y + 1.772(C_b - 128) \quad (6.6)$$

Converting from Greyscale to RGB requires setting each channel to the value of the greyscale channel (e.g. Grey = 128 becomes RGB = (128, 128, 128)). Downgrading from RGB to Greyscale can be performed with the following equation.

$$Greyscale = 0.299R + 0.587G + 0.114B \quad (6.7)$$

Further downgrading from Greyscale to Bitonal can be achieved by setting a threshold halfway along the grey scale. Converting from Bitonal to Greyscale requires changing from 'Bitonal = 0 or 1' to 'Greyscale = 0 or 255' (or vice versa depending on which value corresponds to black).

6.3.4 Transparency

The Alpha (transparent) channel can be represented as another channel in the `Palette`, by setting the `Channel` to `ChannelType.ALPHA`. It can be given a range

such as 0-255 which would represent a PNG channel, or a range of 0-1 representing a GIF channel. Care must be taken in the GIF scenario. There are 256 available colours, to allow transparency one of these colours must be designated transparent, as shown in figure 6.2. When the display system encounters the RGB values of the designated transparent colour, those pixels are not displayed.

6.3.5 Pixel Resolution

This is often specified as an X and Y resolution of the pixel, along with a unit of measurement. Using these two values allow the aspect ratio to be calculated. Printers use very high resolutions, giving high definition output, whereas the display constraints of monitors mean their resolutions are much lower.

6.3.6 Compression

Section 5.3 serves as an introduction to compression techniques. The compression technique used in an image is not a significant property. Ideally we want to store the image in a data file uncompressed as it reduces the chance of losing information, the drawback here lies with the resultant large files. The Migration on Request tool does not convert between compression techniques, rather, decompression is an additional step when parsing the input file and compression when writing a new file.

To allow maximum usage of the Migration on Request tool, it is advisable to use an open standard compression technique. For example, the LZW compression technique is patented by Unisys [44], who have begun demanding royalties for its use. The PKZIP method, developed for UNIX compression tools such as `gzip`, is found in PNG and is a more suitable technique.

Compression and decompression code is contained within separate modules code. While this may result in duplication of code, the code required for each format is often too specific to be modularised.

6.4 Input

It is much more difficult to read in a file than to output one. This is because we know precisely what the intermediate structure holds and can interpret and output accordingly, whereas a data file can have many variations within it, of which we may be dealing with any combination.

6.4.1 PNM

When reading in a PNM file, data is inserted into the `Document` structure as we come across it. The implementation has three boolean fields called `dimensionsSet`, `paletteSet` and `dataSet`. Only when all three have become `true` can we return with the `Document` object.

An infinite `while` loop is entered; each iteration reads a line of the file and stores the result in a string. The continual loop is needed as we cannot predetermine how many lines of comments there may be. As a PNM file always has the same layout, the boolean fields allow us to verify whether we will read in the dimensions, palette or data values next.

Recall that a PBM file has only black and white colour information (section 5.3.1.1): if this file type is encountered the program automatically sets the `Palette` object and `paletteSet` boolean to record this. In all other cases, the `pnmSetDimensions()` method is called first and interprets the string containing the line that has been read, inserting the relevant data into the `Dimensions` object. Assuming we do not encounter any comments on the next iteration of the loop, the `pnmSetPalette()` method interprets the string and inserts the data into the `Palette` object.

When the `pnmSetData()` method is called, the active string is interpreted and the values inserted into the `Data` object. However, on returning from this method the program assumes all the data values have been read in. To counter this, the rest of the data values are read in from within the method. As we stored the dimensions of the image into the `Dimensions` object earlier, we can retrieve these and determine when all the data values have been read in. The dimensions allow us to establish how big the data arrays need to be, and we return when they have been filled. When this has occurred we can return to the parent method, and the parent method can in turn return with the complete `Document` object.

6.5 Output

6.5.1 PNM

It was a lot simpler to write the output module for the PNM family of files. The flexibility of the format allows each pixel value to be placed on a new line; this made writing the file very simple. As the output file is an ASCII text file, it was in this instance relatively simple to retrieve the values and write them out to the file.

6.5.2 PNG

Implementing the PNG module caused many problems. Understanding the structure of the PNG file took a long time to grasp; the implementation also required learning about the Java compression operations which demanded some time to understand. The PNG file is broken up into three compulsory chunks: the header, data and trailer chunks. The header was the first section I implemented. This required some investigation into the use of CRC techniques. However, once this was completed it was relatively straightforward. The trailer was also relatively straightforward, however I encountered significant problems with the data chunk.

Filter algorithms are applied to the data on a scanline by scanline basis; the filters prepare the image data for optimum compression. Understanding the algorithms and techniques used caused significant problems. At the same time, there were problems writing ints out to the bytestream. The combination of these problems caused many problems in the implementation. While a temporary implementation has been achieved, there are several flaws and it is not as extendible as it needs to be.

6.6 Implementation Restrictions

Unfortunately time constraints meant the tool was unable to be fully implemented as intended. Rather than continuing with the implementation making small progress into several sections, I decided to leave the implementation at a stage where it could be picked up at an independent later stage, allowing development to continue from then.

The time constraints arose due to the scale of the project. It is a large project and a significant proportion of the time was dedicated to understanding the issues involved, as documented in the first few chapters. Implementing the tool required an understanding of the graphics formats involved, along with an understanding of the Migration on Request techniques involved to do this. Combined with the digital preservation research, the research and development aspect of the project demanded a lot of time.

A justifiable criticism of the project is the time management aspect. It was not that time was wasted, for all the research and investigation that was performed was valid and important. However, to successfully implement a complete and working tool in such a short timescale the investigation should have been more focused on the implementation objective, rather than combined with that of research.

Migration on Request implies the use of the Unified Process development method-

ology, the more traditional style of incremental design. This was suited to the project as it allows requirements capture and analysis and assists the design in not only realising the requirements, but system testing and documentation as well. For a long term solution, this fitted the goals perfectly and was the best methodology to use. The continued use of this methodology is recommended, as it reduces the chance of future problems.

However, using the Unified Process on a project of this scale does take time. Careful management is required to ensure the project does not miss deadlines. Managers should be aware of and encourage the use of some of the more lightweight “Extreme Programming” methodologies. These include requiring developers to make small releases of the project regularly, ensuring visible progress is made; ensuring the project focus is on validation at all times by writing tests before the software; and implementing a good configuration management tool, allowing anyone to change anyone else’s code modules without permission if they believe it is for the good of the system.

A combination of the two methodologies would ensure the design principles were present, while encouraging a rapid release and implementation cycle.

6.7 Summary

This chapter has explained the implementation processes, issues and decisions that were made. Significant progress was made into the implementation of the Migration on Request tool. The decision was taken to leave the tool at a point where implementation could easily continue, rather than making limited progress into several modules. Frameworks for testing the tool are presented in the following chapter.

7. Testing

Practical testing was performed to a restricted level as the implementation was limited. It is, however, possible to detail testing frameworks that could be used in a complete situation.

It is important to ask “What am I testing?” There are several aspects we aim to explore. The first two are very closely linked. Verification of the system asks “have we got the system right?”, validation of the system asks “have we got the right system?” [39]. The third aspect is to test the robustness of the Migration on Request concept, achieved by performing tests upon the system.

7.1 Verification

Verification requires the system to be correct (fault free) and consistent (everything working in harmony). Some methods for testing the system have been described below.

- White box testing requires we analyse the internal structure of the code to determine a flow graph (a series of `if`, `while` and `for` statements). We then follow all these paths, ensuring every statement has been executed and every branch has been followed. This prevents deadlocks and unexpected data flows.
- Integration testing looks for errors in interfaces between components, for example type and range errors, such as the conversion of a `double` to an `int`, or representation errors, such as a length variable exported in millimetres and imported in metres.
- Stress Analysis tests the resource utilisation of the system. First we identify what needs to be stressed (for example the ability of the intermediate structure to handle very large files), then build stress upon it by generating large volumes of data. Another test here would be to test an output modules ability to break a large file into the correct number of data chunks to be written to a file.
- The Migration on Request tool should be tested for ‘Necessity’, which states if things are present which need not be, the structures will be more complex and less understandable. At the same time, the system needs to be ‘Sufficient’. This states we need to ensure that everything required to allow extensions and understanding is present.

7.2 Validation

Validating the system is often more complex than verification. In this situation we need to prove we can access and view the output document, and prove the output document has remained authentic. Accessing and viewing the document can be tested by rendering the image in a standalone application; the following section presents a technique for verifying the authenticity.

7.2.1 Reversible Migration

Although it was not possible to perform reversible migration tests within this project, reversing a migration and comparing the original and output files is the only true way of proving a file has not been altered. This is achievable when using a Migration on Request tool as the integrity is expected to be much higher than that of a regular migration process. Reversing a regular migration would require tools converting back one format at a time, demanding a lot of development work. However, different methods of representation within a file and the loss of information during transformation make successful reversible migration a difficult goal to obtain.

Peter Buneman has suggested, in unpublished work, that we may see developments where the requirements for a successful reversible migration change. Syntactic reversible migration will demand that the files are identical; a very difficult goal to obtain. Semantic reversible migration demands that the meaning of the files are the same. While this allows some flexibility in the conversions migration can perform, the boundaries of the semantics will have to be carefully defined.

Although breaking the single migration model of Migration on Request (as it requires a migration forwards, and then a migration back), input and output modules could be written at the same time. For example, when a new format becomes available, writing an output module allows a procedural comparison with the converted files from the previous output module, ensuring we can continually render the files. Writing the input module takes immediate positive steps to preserve files of that type. Having both input and output modules for a format allows a direct conversion from format A directly back to format A, allowing us to verify the correctness of the modules.

Writing and preserving an output module for an obsolete format may be invaluable to future generations. If a current format can be downgraded and converted back into an obsolete format, we can test tools such as emulators by comparing the (emulators) output of the downgraded file with a rendering of the original file from a current application. Conversely, emulators can assist the development of

migration tools, as we can render the original file in the emulator for comparison with the output of the migration tool.

7.3 Robustness

Migration on Request is intended to produce a portable system to be used across a long period of time. Testing the robustness of the concept requires several different aspects of an implementation to be tested.

By producing a program and ensuring it works correctly, we have already verified and validated the system on a certain platform. It is now necessary to test the system across different systems, ensuring the code that has been written is portable. This is likely to be successful as the program runs through the Java Virtual Machine rather than directly on the hardware platform. Sun's motto on Java is "Write once, Run anywhere"; they have performed a lot of tests to integrate Java into many systems and raise its level of support, trying to make Java ubiquitous.

Testing the system with new input and output modules is important, and not just for the system. The documentation provided needs to be tested, as future developers need to be able to clearly interpret and follow the instructions provided to them. A test here could be to ask other developers to implement a module requiring alterations to the intermediate structure. This would test the instructions and explanations provided, while ensuring the system allows expansion.

Verifying the longevity of the solution is a difficult test to perform, and one which may well not be answered until it's too late. However, the Migration on Request technique can be tested across the ages to a certain extent. An implementation should be compiled and (made to) run on machines from as early as possible. For example, I have available to me a working Apple Macintosh Classic II [29], released on October 21st 1981. This machine had a Motorola 16Mhz 68030 processor, shipped with a 40MB hard disk and 2MB RAM, allowing a maximum of 10MB. The speed and space increase since then implies we do not have to worry about efficiency and size constraints of the program, but should focus on the design and readability.

Once the program is working on the machines of yesterday (for example, a C program on the Macintosh Classic II), we should then attempt to compile and run the programs on today's machines. This may highlight long term portability and implementation issues and give an indication of where future problems may lie. While we cannot predict what developments may be made in the future this may assist us to find a heuristic evaluation.

7.3.1 Multiple Migrations

Producing, compiling and running a tool on a machine of yesterday and porting it to a machine of today allows us to investigate the detrimental affects a series of migrations will cause in comparison to the (expected) minimal change from a Migration on Request tool.

To perform this test, the chain of migrations that a file would have taken over the ages should be looked at and then performed. In the raster graphic scenario, a PNM image may have been converted into a BMP, then a JPEG, then finally a TIFF image. A direct conversion from a PNM to a TIFF using the tool would allow verifications to be made.

7.4 Summary

This chapter has presented a number of frameworks to test a Migration on Request tool. The concept of Migration on Request is a developing one; by performing tests based on and related to those presented here the development may be stimulated.

8. Evaluation

8.1 Research Considerations

It was surprisingly difficult to find concise and useful literature on the topics covered. Digital Preservation is a newly emerging topic; much research is being published in this field. At the moment it is difficult to separate the essential documents from those simply rewording other work. This is understandable as many people in this field come from an archiving background and have little technical knowledge; by reading several descriptions of a process it can often assist our understanding.

Collecting information on specific graphic formats also caused difficulties, this may have severe consequences. Without the specification to assist our interpretation of a format, it may be near impossible to access a format for conversion. Steps need to be taken to secure the specification of proprietary formats; sadly many vendors are currently unwilling to assist this process. However, discussions have taken place and proposals made to set up a long term file format archive which would be invaluable to future generations [1].

Current format registries such as Wotsit [36] only provide links to specifications, fortunately the government is beginning to take steps with the PRONOM database system [35]. It is essential to ensure the persistence of specifications and in a perfect world, supplement these with descriptive information in layman's terms. The latter (and perhaps over-the-top) scenario would ensure ambiguous specifications were avoided.

8.2 Evaluation of Intermediate Structure

8.2.1 Comparison with Java Image Class

The `Java.awt.image` package contains classes and interfaces for manipulating images. Prior to Java 1.2, working with images was tricky as the image processing model was based on streaming image data being loaded across a network, making images easy to display while loading but difficult to manipulate [15]. Java 2D extends the model to a comprehensive graphics system.

A `BufferedImage` represents an image as a rectangular array of pixels using a `Raster` object and a `ColorModel` object capable of interpreting the `Raster`'s

pixel values. A `BufferedImage` holds the image data in memory, allowing ease of manipulation.

A `Raster` object is composed of a `DataBuffer` that contains raw pixel data and a matching `SampleModel` that knows how to extract pixel data from the `DataBuffer`. Most applications simply use `BufferedImage` objects and never have to work with `Raster` objects directly. A `WritableRaster` object is required to allow direct modifications to the pixel values.

The `ColorModel` object extracts individual colour components from pixel values. The default ARGB colour model packs 8-bit colour and alpha components into a 32-bit `int` in `0xAARRGGBB` format. The colour space can be defined for other spaces too, such as the CMYK space.

Pixel values, stored in the `DataBuffer` of a `Raster`, do not necessarily fit into `int` values, so `ColorModel` methods extracting colour components come in two forms. In the first, the pixel is specified as an `int`. In the second, the pixel is an `Object`, comprising an array of primitive values. This primitive type is selected from a set of constants and is specified in the `ColorModel`.

The supplied methods allow a significant degree of manipulation on the image; methods are also present to display, scale and create composite images. The functionality of Java 2D far supersedes that of the implemented Migration on Request tool, as Java 2D is an image processing model.

The `Image` object itself contains the dimension information. This appears limited to the width and height of the image; there are no fields for the pixel resolution. The Java 2D package was never intended as a conversion package; including these fields in the Migration on Request tool was an important addition.

The `ColorModel` object is well designed and thorough. However, the lack of storage facilities for a palette makes it unsuitable for use in a Migration on Request tool. There also appears to be little differentiating between the varying colour depths possible.

The structure for storing the data is well defined; of particular note is the `SampleModel` object. The variety of storage systems for data values (e.g. a packed 24- or 32-bit `int`) makes this a valuable inclusion. The Migration on Request tool has no comparison as each value is stored individually in an array. However, as values can have variable length, a redesign may include an explicit field recording these lengths. Allowing values to be stored within a packed `int` or in an array allows extendibility. Comparitively the Migration on Request tool allows us increase the size of the array if more channels are developed.

The methods for retrieving values from the `Raster` object are flexible, similar to those in the Migration on Request tool. The default storage system of image

data is from the top-left, although this can be user defined. Again, the Migration on Request tool allows similar flexibility.

The Java 2D structure is a powerful display system, the purpose for which it was designed. As a Migration on Request tool, it lacks some required features and extensibility. Many algorithms and methods are hidden to the application programmer; this is undesirable as it may make the code more difficult for future developers to understand. However, with some modifications it could become a suitable intermediate structure. The support given to a Java-released package would help ensure the longevity of a solution.

8.2.2 Duplication

In retrospect, there are a number of flaws with the finished design of the intermediate structure. Although not critically damaging, they do result in some duplication of code.

In the `Dimensions` object, there is both a `width` and `height` variable, yet there is also an `imageSize` variable. This is calculated by multiplying the width with the height; note it is different from the `screenSize` variable. The `imageSize` variable could easily be removed, but the question remains as to why it was included originally. A number of image formats include within their dimensions information the image size. Although it was clear that this was derived from the width and the height, it was included simply to correlate between the original formats and the intermediate structure. Care should be taken by future developers. As discussed in section 7.1, always restrict the structure to the minimal set required.

The `Page` object included the `Palette`, `Dimensions` and `Data` objects. This was designed to allow multiple images within one file. However, the dimensions of each image within a file are likely to be the same. GIF allows Local Colour Tables to override the Global Colour Table, so the option to have different `Palette`'s must be included. At some stage in the future, these images may have different dimensions; currently this seems unlikely and the duplication of objects seems unnecessary. However, raising the `Dimensions` object to the `Document` object will result in a less intuitive design. In this instance, the duplication is beneficial.

8.2.3 Range of Features

The tool has one significant omission from the range of features found in raster graphics: layers. These allow composite images to be composed and text to be inserted, moved and altered. To include layers, the intermediate structure may have to be altered. The `Page` object could be modified to resemble a bounding

box, with coordinates showing where this bounding box should be placed. Regular images could fill a bounding box to the edges. However, this design may begin to resemble a vector graphic tool, as the boundaries between the two are blurred. If the crossover is too great, perhaps the graphic tool should include both vector and raster graphics.

8.2.4 Number Systems

A significant issue with the design of the tool lies with the incorrect focus on the underlying number systems used. An insufficient investigation was not conducted on the length of `int`'s (e.g. 16- or 32-bit) until a late stage. We may see developments made to number systems over the next few decades, if not sooner. These may result in incorrect number lengths, consequently there may be pointers to incorrect places causing serious problems for the program. An initial lack of understanding of number systems meant this critical area was initially neglected.

The development cycle became focused upon implementing almost anything to establish what was needed next. This method, known as clearing the fog, is often thought of as a dangerous software engineering practice as it can be easy to avoid the problems you set out to find. Here, it was easy to avoid the issues with number systems until a late stage. Accounting for this sooner may have removed some of the output module implementation issues.

8.2.5 Value Class

The `Value` class wraps `int`'s and `double`'s, preventing changes to number systems affecting the entire program. The `Value` class that was implemented was cumbersome and difficult to use. Manipulating the values within the class was not as easy as the primitive Java types. To rectify this, the methods should be studied and corrected. This requires an investigation into the methods that can be performed on both `int`'s and `double`'s, and knowledge of the methods that can be performed upon Java classes such as `Integer`.

8.2.6 Big and Little Endian

The intermediate structure has no method of determining whether a file has been parsed in big- or little-endian format. This is a serious exclusion; a redesign should include a flag noting which we are operating in and provide methods to convert between the two.

8.3 Portability and Longevity

8.3.1 Coding Style and Comments

To assist further development, it was important to follow a clear, regulated coding style throughout. Following the conventions was achieved with little problem. It is, in fact, easier to follow a clear coding style than to not follow a style. Clear tabs allow quick verification of where a method ends, following naming conventions often makes the purpose of a method clearer.

Comments were more difficult to continue throughout coding, although it was often simple to correct this. While it is possible for Java developers to understand the code without the comments, a future developer may find understanding the code a lot harder, particularly if the coding paradigm has shifted. Each method has a small description associated to it; this gives a human readable meaning to each method explaining the algorithms involved in coding.

8.3.2 Java--

The development of a subset of Java is a significant and daunting task. Within this project, the majority of the work could be completed with basic operations. This made developing and justifying a subset of Java very difficult, simply because the inclusion of many classes was not required.

The intermediate structure only required the inclusion of the `Vector` class in the `Document` object. This class was used as it is not possible to predetermine the number of comments that were to be found when the document was parsed.

The PNM input module required the use of `java.util.StringTokenizer`. This was included as PNM data consists of rows of numerical values separated by spaces. This class is an important class to the Java package, and is one expected to last into the future. Like all the classes that were written requiring the inclusion of special packages and Java classes, there are clear comments recording they have been used.

Justification for inclusion required looking at the classes and estimating their longevity. It is wise to use the highest class possible in a package hierarchy. An estimation of the importance of the class also assists justification. For example, the `java.util.zip.Deflater` class will be commonly used not only to compress image data, but also to create standard compressed files.

It would be tedious and excessive to expect programmers to comment each line of code that used an external class. Rather, each class should have a header recording which external classes are used and in which methods. This, combined

with the Java documentation that is expected to remain available into the future should assist developers to a comfortable level.

8.4 Implementation

8.4.1 Programming Experience

Undertaking this project allowed deep insight into what was involved in a major programming project. The degree course provides a firm foundation in programming, however applying this foundation to a project of this magnitude required a significant amount of time learning about a number of the techniques needed.

The Java Image class has spawned a number of decoders and encoders for a variety of raster formats. As an insight into how to attempt the code, and due to the rapid development cycle of the project, a number of these packages were used as a reference for the coding. These gave me advice on enhancing the readability of the code, as I could develop opinions of my own on the attractive and flawed aspects that were found.

The open source ImageProperties class [41] provided help on extracting information from an image; likewise the PngEncoder class [13] assisted me with the compression techniques for the PNG image. Although these eased the development, questions about reuse are raised: in a long term scenario, should code reuse be allowed? The methods and techniques which were learnt from this code may now be outdated and cause more problems than would otherwise have occurred.

Over the relatively short period of time we are discussing, this is unlikely to be a problem. In this instance comparisons were made with the official Sun specification [42] to verify the code in question was not outdated. However, over a longer period of time with a less stable language, this may be a greater problem.

8.4.2 Resource Utilisation

The `Value` class caused several difficulties when creating the program. Manipulating the values within the object was difficult, this caused me to create an excessively large number of `new Values` through the program. When these inefficient techniques were used within the `Data` object, it caused the program to crash when dealing with large files. To rectify this, the methods for manipulating `Values` were extended and the code was in turn modified to reduce the amount of variables being created.

8.4.3 Number System Issues

When writing the PNG output module, difficulties were encountered when writing values to the bytestream. These were overcome by further investigation into the number systems used in the PNG format. The open nature of the PNG standard meant these were simple to obtain; discovering the number systems used in proprietary formats may be more difficult. It is necessary to reiterate the importance of a thorough understanding in this field.

8.4.4 Implementation Output

Section 6.6 details the reasons for restricting the implementation. However, it is interesting to look at the output that can be obtained from the tool. Both input and output modules for the PNM format were implemented. These work successfully; the output file is unchanged from the file that is input. This proves that the intermediate structure is working properly, and in this instance the integrity of the output file is complete.

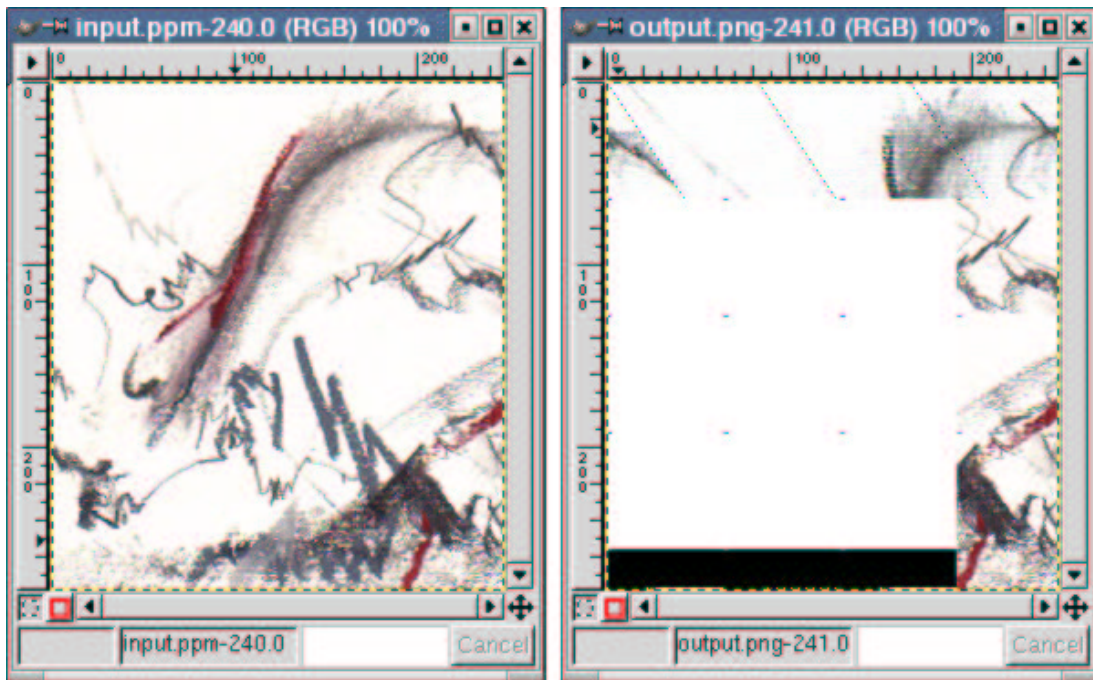


Figure 8.1: Comparison of PNM input and PNG output

Figure 8.1 shows a PNM input file and the resultant PNG output file. Clearly, this is not working as it should be, however it is also possible to see that it is nearly complete. The problem appears to be with the filter bytes that complete

each row; these are currently incorrect, causing the twisting effect that can be seen in the top third of the image.

The lower right quarter of the image appears to have been successfully converted. Obviously the entire image needs to be displayed correctly. While testing needs to occur to confirm the integrity, the visual check implies a working PNG module would also produce a valid output file.

Despite the output not working as it should be, it is possible to see the intermediate structure again is not causing problems (as the size of the image is correct and some data has been successfully converted). Rather than a conceptual design problem, it is a programming problem that is yet to be solved.

At the moment, it is not possible to compare the Migration on Request tool to commercial conversion packages. Of course, this project was never intended to be used as a straight up conversion package. The long term methodologies used differentiate it the many packages available. However, the PNG output, although incomplete, indicates the output from a complete package would be comparable to that of a regular conversion package.

8.5 Summary

This chapter has provided an evaluation of the issues faced in the development and implementation of a Migration on Request tool. Improvements were suggested for several features. Suggestions were made for the most suitable method of continuing the project.

9. Conclusions

9.1 Original Evidence

In developing the tool, the most useful test for correctness was to simply compare the original and converted file using a standalone graphics application. Without a method of comparison, verifying the correctness of the tool would have been much harder. It is important to write tools now while comparisons can still be performed. Evidence of correct output should also be produced now and preserved; this will assist verification at a later date. This evidence can come in a number of forms; common suggestions include screenshots and written descriptive documents.

9.2 File Formats

The research and development of preservation techniques and tools can be expected to continue indefinitely. A significant quantity of this research should be invested into the file formats themselves. The suggestion of collecting specifications and documentation has already been made and preparations for these are occurring. Physical libraries and archives should be encouraged to participate in these, preventing the specifications themselves from being unreadable.

Studies should investigate the directions the formats themselves are heading. For example, an indication of the colour spaces we can expect in the future may assist the development of a more successful raster graphic tool.

As technology evolves, so does the formats. The question of what happens in this situation is an important one. For example, Computer Aided Design (CAD) formats are influenced by developments in the Virtual Reality (VR) world. If technology enhancements allow VR (and thus CAD files) to develop a format that is completely different to today's format, should we migrate the file? For example, suppose in a future CAD format we can "walk" around, pick up objects and develop the file interactively. Should we migrate a CAD file that consists only of 3 perspective views into this? If the original file is so basic we have to fill in a large number of arbitrary values, can we assume we have not influenced the resultant view?

Following open standards may give some guidelines as to where a format is heading; in many situations it may be wise to base the intermediate structure upon

an open standard. Note though, a format is not necessarily a good format just because it is standardised.

9.3 Intermediate Structure Design

Designing the intermediate structure is not a task to be undertaken lightly. Achieving a design that is modular and extendible will require careful planning and thought. The design may be influenced by expected future developments and designed to allow the incorporation of alternative developments, however it is always based on the file formats that have appeared to the present day. It is important to understand that the designer can only be expected to do their best. At some stage the developments may surpass the abilities of the design, requiring a new structure incorporating features both old and new. Clearly defined interfaces between input and output modules will assist this redevelopment process.

9.4 Languages

The object-oriented nature of Java made constructing the intermediate structure a logically simple task. Unfortunately, changes in the language may make understanding the code very difficult even after ten years. However, the majority of the code was written using the fundamental operations of the language and the quantity of published literature on Java indicates definitions will be available for the other classes for a long time. Even if the language drastically changes, documentation should provide developers with help in understanding it.

Recent discussions suggest Java is already evolving for its next release [25]. Paul Graham suggests Java will not last into the future [37]; however he raises some interesting (and controversial) points about language evolution. The last fifty years have seen languages evolve slowly. This is because languages are not technologies susceptible to leaps of advancement. Rather, they are a formal description of a problem you want a computer to solve. He suggests this notation should be made up of a set of fundamental operators, or axioms, and we should be able to write the rest of the language from these.

Therefore, we seem on the right lines to write a Migration on Request tool using the minimal and most basic set of operations available. The fundamental set of operators of the future may share some resemblance to the set of today, and is more likely to share this minimal set than a full set of complex Java classes.

9.5 Authenticity

Verifying the integrity and authenticity of a digital object requires evidence that the methods used to store and access the object adhere to a comprehensive series of management and computing protocols. Currently, many of these long term protocols have yet to be determined and established. This is an important field and is expected to influence the practical techniques used to maintain access to digital objects.

9.6 Reversability

Reversible migration is an important method of verification and validation. The differences and boundaries between a semantic and a syntactic reversal are yet to be established, but may prove essential to achieve reversible migrations. Preserving obsolete output formats may allow emulators to be tested, assisting other techniques of digital preservation in their development.

9.7 Future Developments

The latter half of this document details frameworks and possible developments that may occur within the field of digital preservation. These are intended to assist future developers in developing and testing a Migration on Request tool.

This implementation has been left at a stage where it can easily be continued. The next step will be to implement more input and output modules and test the conversion routines. QuickTake would be the most desirable input format to implement, as many real world users are currently unable to access the format, making the tool instantly usable. This is beneficial as a tool under demand is more likely to be maintained. If the tool was to be used in such a way, the focus on longevity and portability should not be lost.

Once a tool has been implemented, the frameworks as detailed in chapter 7 should be utilised. This will further investigate the portability and longevity issues of the Migration on Request technique.

9.8 Conclusion

Migration on Request is a valuable approach to Digital Preservation. In some form, it looks to be an important technique as a preservation method. However,

considerable research needs to be conducted before it can be used as a successful practice, covering factors such as the intermediate structures and where these may be heading; language, authenticity and integrity considerations; and development protocols ensuring the longevity and portability of current projects.

This implementation has only touched upon many of the features that should be included in a complete Migration on Request tool. In a real world situation, the consequences of an incomplete, underdeveloped tool cannot be underestimated. If this document can assist others to develop a complete tool by highlighting areas of importance, then I consider this project to be a success.

Bibliography

- [1] Stephen Abrams. Proposal for a Format Registry for Digital Library Preservation, http://hul.harvard.edu/~stephen/Format_Resistry.doc, 2002.
- [2] Adobe Developers Association. *TIFF Revision 6.0*, June 1992.
- [3] Apple Computer, Inc. QuickTake 150: Specifications (9/96), <http://docs.info.apple.com/article.html?artnum=17410>, 1995.
- [4] Apple Computer, Inc. *Mac OS X: Unable to Open QuickTake 100, 150 Pictures, Article ID: 61047*, January 2002.
- [5] BBC News. Fire Destroys Librarian's Work, <http://news.bbc.co.uk/1/hi/scotland/2558655.stm>, 2002.
- [6] David Bearman and Jennifer Trant. *Authenticity of Digital Resources: Towards a Statement of Requirements in the Research Process*, June 1998.
- [7] Testbed Digitale Bewaring. *Digital Preservation Test Bed White Paper: Migration - Context and Current Status*, December 2001.
- [8] CAMiLEON. BBC Domesday, <http://www.si.umich.edu/CAMiLEON/domesday/domesday.html>, 2003.
- [9] CAMiLEON Team. CAMiLEON, <http://www.si.umich.edu/CAMiLEON/>, 2002.
- [10] CompuServe Inc. *GRAPHICS INTERCHANGE FORMAT(sm), Version 89a*, July 1990.
- [11] Derek A. Benner. Learn Graphics File Programming with Delphi 3, <http://kbt.narod.ru/docs/lgfp/ewtoc.html>, 2001.
- [12] Digitale Duurzaamheid. The Virtual Library of the Digital Preservation Testbed, <http://www.digitaleduurzaamheid.nl/index.cfm?paginakeuze=185&categorie=%2>, 2003.
- [13] J. David Eisenberg. PngEncoder - convert a Java Image to PNG, <http://catcode.com/pngencoder/>, 2000.
- [14] Thomas Boutell *et al.* *PNG (Portable Network Graphics) Specification, Version 1.0*, October 1996.
- [15] David Flanagan. *Java Foundation Classes In A Nutshell*. O'Reilly and Associates, Inc, 1999.

- [16] Consultative Committee for Space Data Systems. *Reference Model for an Open Archival Information System (OAIS)*, January 2002.
- [17] Eric Hamilton. *JPEG File Interchange Format, Version 1.02*, September 1992.
- [18] Margaret Hedstrom and Christopher Lee. *Digital Objects: Definitions, Applications, Implications*, May 2002.
- [19] David Holdsworth. *C-ing ahead for Digital Longevity*, August 2001.
- [20] David Holdsworth and Derek Sergeant. *A Blueprint for Representation Information in the OAIS model*, March 2000.
- [21] David Holdsworth and Paul Wheatley. *Emulation, Preservation and Abstraction*. CAMiLEON, August 2001.
- [22] Scott Hommel. Code Conventions for the Java Programming Language, <http://java.sun.com/docs/codeconv/>, 2003.
- [23] interPARES Project. *Preservation Task Force Report*, June 1999.
- [24] interPARES Project. *Authenticity Task Force. Appendix 2: Requirements for Assessing and Maintaining the Authenticity of Electronic Records*, March 2002.
- [25] Janice J. Heiss. New Language Features for Ease of Development in the Java 2 Platform, Standard Edition 1.5: A Conversation with Joshua Bloch, http://java.sun.com/features/2003/05/bloch_qa.html, May 2003.
- [26] Jef Poskanzer. PBM, PGM, PNM, and PPM, <http://netghost.narod.ru/gff/graphics/summary/pbm.htm>, 1994.
- [27] JPEG. JPEG 2000: Links to Information, <http://www.jpeg.org/JPEG2000.htm>, 2003.
- [28] KB. Koninklijke Bibliotheek, <http://www.kb.nl/index-en.html>, 2003.
- [29] Brock Kyle. Apple Macintosh Classic II Specs, http://www.everymac.com/systems/apple/mac_classic/stats/mac_classic_ii.%html, 2001.
- [30] Raymond A. Lorie. *Long-Term Archiving of Digital Information*, June 1999.
- [31] Clifford Lynch. *Canonicalization: A Fundamental Tool to Facilitate Preservation and Management of Digital Information*, September 1999.
- [32] Clifford Lynch. *Authenticity and Integrity in the Digital Environment: An Exploratory Analysis of the Central Role of Trust*, June 2000.

- [33] Phil Mellor. *Thoughts on Using C- in the Practical Implementation of a Vector Graphics Migration on Request tool*, December 2002.
- [34] Phil Mellor, Paul Wheatley, and Derek Sergeant. *Migration on Request - a Practical Technique for Preservation*, June 2002.
- [35] Public Records Office. PRONOM File Format Database, <http://www.pro.gov.uk/about/preservation/digital/pronom/default.htm>, 2003.
- [36] Paul Oliver. Wotsit: The Programmers File Format Collection, <http://www.wotsit.org>, 2002.
- [37] Paul Graham. The Hundred Year Language, <http://www.paulgraham.com/hundred.html>, 2003.
- [38] RLG-OCLC. *Attributes of a Trusted Digital Repository: Meeting the Needs of Research Resources*, August 2001.
- [39] Dave Robertson. *SEOC 2: Verification and Validation*. Edinburgh University, February 2003.
- [40] Jeff Rothenberg. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*. Council on Library & Information Resources, 1999.
- [41] Marco Schmidt. ImageInfo - Java Class to Extract Image Properties, <http://www.geocities.com/marcoschmidt.geo/image-info.html>, 2003.
- [42] Sun Microsystems, Inc. Java 2 Platform, Standard Edition, v1.4.1, API Specification, <http://java.sun.com/j2se/1.4.1/docs/api/index.html>, 2003.
- [43] TechTarget. Big-Endian and Little-Endian: A Whatis.com Definition, http://whatis.techtarget.com/definition/0,,sid9_gci211659,00.html, 2001.
- [44] Unisys. Unisys — LZW Patent and Software Information, http://www.unisys.com/about__unisys/lzw, 2003.
- [45] USA Today. Digital Memory Threatened as File Formats Evolve, http://www.usatoday.com/tech/news/2003-01-17-digital_x.htm, 2003.
- [46] Paul Wheatley. *The CAMiLEON Guide to New Digital Preservation Strategies*. CAMiLEON, April 2003.
- [47] Paul Wheatley. *CAMiLEON Guide to the Relative Costs and Merits of Different Digital Preservation Strategies*, May 2003.