The International Journal of Digital Curation
Issue 1, Volume 5 | 2010

# A Framework for Software Preservation

Brian Matthews, Arif Shaon, Juan Bicarregui and Catherine Jones,
e-Science Centre, Science and Technology Facilities Council,
Rutherford Appleton Laboratory, Oxon, UK

**Abstract**

Software preservation has not had detailed consideration as a research topic or in practical application. In this paper, we present a conceptual framework to capture and organise the main notions of software preservation, which are required for a coherent and comprehensive approach. This framework has three main aspects. Firstly a discussion of what it means to preserve software via a *performance model* which considers how a software artefact can be rebuilt from preserved components and can then be seen to be representative of the original software product. Secondly the development of a model of software artefacts, describing the basic components of all software, loosely based on the FRBR model for representing digital artefacts and their history within a library context. Finally, the definition and categorisation of the properties of software artefacts which are required to ensure that the software product has been adequately preserved. These are broken down into a number of categories and related to the concepts defined in the OAIS standard. We also discuss our experience of recording these preservation properties for a number of BADC software products, which arose from a series of case studies conducted to evaluate the software preservation framework, and also briefly describe the SPEQS toolkit, a tool to capture software preservation properties within a software development.[1]

[1] This paper is based on the paper given by the authors at the 5th International Digital Curation Conference, December 2009; received November 2009, published June 2010.
The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. ISSN: 1746-8256 The IJDC is published by UKOLN at the University of Bath and is a publication of the Digital Curation Centre.

## Introduction

Software is a class of electronic object which is frequently the result of research and is often a vital pre-requisite to the preservation of other electronic objects. However, there has only been limited consideration of the preservation of software as a digital object in its own right. This is mainly owing to the inherent complexity of software products - a typical software artefact has a large number of components related in a dependency graph, with specification, source and binary components, and a highly sensitive dependency on the operating environment. Handling this complexity is a major barrier to the preservation of software, especially for people who were not involved in its development but nevertheless want to maintain access to software. Further, the preservation of software is frequently seen as a secondary activity and one with limited usefulness.

Software preservation is thus a relatively underexplored topic of research and there is little practical experience in the field of software preservation as such. Given the relative immaturity of the field, there is a need for both a conceptual analysis of the process of software preservation, and experience and tools in undertaking preservation in practice.

The work presented in this paper represents a pair of studies into software preservation[2], which looked at a number of software repositories and other groups engaged in maintaining software over the long term (Matthews, McIlwrath, Giaretta, and Conway, 2008). As part of this study, we have developed a framework to express the notion of software preservation and set out some baseline concepts of what it means to preserve software. The framework also develops and extends the notion of performance and emphasises the notion of adequacy and relates it to authenticity; narrows the notion of significant property to those properties which are testable within a performance. Additionally, it considers the concepts introduced within the Open Archival Information System (OAIS) reference model (Consultative Committee for Space Data Systems (CCSDS), 2002), and uses them within the framework to categorise the preservation properties identified within the model. In our study, this framework was developed in conjunction with a number of analyses into software preservation practice, and some experimental tool development. In this paper we concentrate on the conceptual framework. We briefly discuss the exemplars and tools. For further details on the methodological and exemplar results of the study see Matthews, Shaon, Bicarregui, Jones and Woodcock (2009a, 2009b).

## Aspects of Software Preservation

Long-term preservation of software has the following four major aspects:

- **Storage**. A copy of a software "product" needs to be stored for long-term preservation. As a software product is a complex digital object with potentially a large number of components, what is actually preserved is dependent on the software preservation approach taken. Whatever the exact items stored, there should be a strategy to ensure that the storage is secure and maintains its authenticity over time, with appropriate strategies for storage replication, media refresh, format migration etc. as necessary.

[2] Joint Information Systems Committee (JISC) study into the *Significant Properties of Software* (2007): http://www.jisc.ac.uk/whatwedo/programmes/preservation/2008sigprops.aspx and Project Tools and Guidelines for Preserving and Accessing Software Research Outputs (2007-9): http://www.jisc.ac.uk/whatwedo/programmes/reppres/tools/software.aspx

- **Retrieval**. In order for a preserved software product to be retrieved at a future date, it needs to be clearly labelled and identified, within a suitable catalogue. This should provide search on its function (e.g. terms from controlled vocabulary or functional description) and origin.
- **Reconstruction**. The preserved product can be reinstalled or rebuilt within a sufficiently close environment to the original that it will execute satisfactorily. For software, this is a particularly complex operation, as there are a large number of contextual dependencies to the software execution environment which are required to be satisfied before the software will execute at all.
- **Replay**. In order to be useful at a later date, software needs to be replayed, or executed, and perform in a manner which is sufficient close in its behaviour to the original. As with reconstruction, there may be environmental factors which may influence whether the software delivers a satisfactory level of performance.

While other digital objects also require these aspects, for software, reconstruction and replay are particular concerns, as more than other objects we are more interested in what software *does* than what software *is*.

## Performance Model and Adequacy

Given the uncertainty of long-term digital preservation, it is necessary to be able to measure the effectiveness of a digital preservation strategy. In the case of software we propose to base this on the notion of how a sufficient level of *performance* preserves the required characteristics of software. Performance as a model for the preservation of digital objects was defined by the National Archives of Australia (NAA) (Heslop, Davis, & Wilson, 2002) to measure the effectiveness of a digital preservation strategy. Noting that for digital content, technology (e.g. media, hardware, software) has to be applied to data to render it intelligible to a user, they define a model as in Figure 1. Here *Source data* has a *Process* applied to it (in the case of digital data some combination of hardware and software) to generate a *Performance*, where meaning is extracted by a user. Different processes applied to a source may produce different performances. However, it is the properties of the performance which need to be considered and can arise from a combination of the properties of the source with the technology applied in the processing.

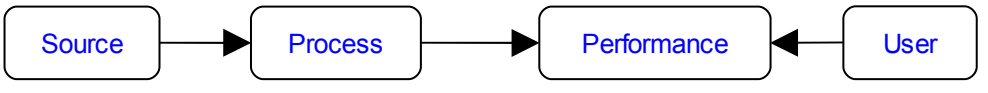


Figure 1. NAA Performance Model.

In general, the performance of a software product is the execution of the binary files associated with the product on some hardware platform configured in some architecture to provide the end experience for the user. However, the processing stage depends on the nature of the software artefacts preserved which have differing reconstruction and replay requirements. For example, in the case where binary is preserved, the process generating the performance requires the original operating software environment and possibly the hardware too, or else emulating that software environment on a new platform. In this case, the emphasis is usually on performing as closely as possible to the original. On the other hand, when source code and configuration and build scripts are preserved, then a rebuild process can be undertaken,

using later compilers and linkers on a new platform, with new versions of libraries and operating systems. In this case, we would expect that the performance would not necessarily preserve all the properties of the original (e.g. systems performance or exact look and feel of the user interface), but have some deviations from the original.

Thus, a software performance can result in some properties being preserved and others deviating from the original or even being disregarded altogether. Therefore, in order to determine the value of a particular performance, we define a notion of *Adequacy*.

*A software product (or indeed any digital object) can be said to perform* adequately *relative to a particular set of features ("significant properties"), if in a particular performance (that is after it has been subjected to a particular process) it preserves that set of significant properties to an acceptable tolerance.*

This notion of adequacy is usually viewed as an aspect of the established notion of *Authenticity* of preservation (i.e. that the digital object can be identified and assured to be the object as originally archived). However, we feel that it is useful to separate these two notions in order to establish a more lucid requirement specification of long-term preservation of software. For this, we use the premise that the term *Authenticity* in long-term preservation essentially signifies the level of *trust* between a preserved software product and its future end users. From the perspective of an end user of a software product, this trust is primarily associated with the ability to trace the provenance and verify the fixity information of the software. For example, a preserved software product with comprehensively documented provenance history, including history of original and custodianship record, and verifiable fixity information, through the use of checksums, might establish in its users a sense of trust of the body responsible for its preservation. But this "trusted preservation" does not guarantee a reliable behaviour from the software once reconstructed in future; it might incur a loss of some of its original features during its reconstruction process. However, the software could still be used for the remaining features retained after reconstruction, which could be sufficient to extract an acceptable level of performance from the software. An example of such software is the emulated version of the 1990's DOS-based computer game Prince of Persia[3]. While some of the operations do not always work on the emulator and the original appearance of the game is also somewhat lost, it is possible to run the emulator to play the complete game on a contemporary computer platform. The term *Adequacy* introduced here is intended to represent this particular concept. As we shall see below, by measuring the adequacy of a particular performance of a software product, we can thus determine how well the software has been preserved and replayed.

***Performance of software and of data***

A further aspect of the notion of software performance is that the measure of adequacy of the software is closely related to the performance of its input *data*. Moreover, the purpose of software is (usually) to process data, so the *performance* of a software product becomes the *processing* of its input data. Thus, applying the NAA performance model to software, we illustrate the relationship between software and its input data as in Figure 2. Note that we have reversed the arrow between performance and user to reflect the information flow. Further, there is an interaction between the

[3] Best Old Games | Prince of Persia Download: http://www.bestoldgames.net/eng/old-games/prince-of-persia.php

user and the software performance, reflecting the user's interaction with the software product during execution, changing the data processing and thus its performance.

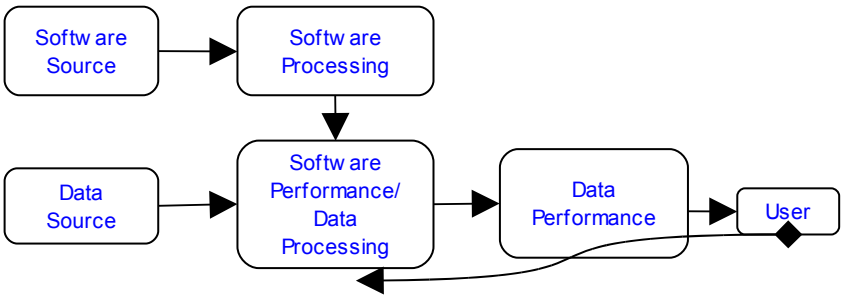


Figure 2. Performance model of software and its input data.

So for example, in the case of a word processing product which is preserved in a binary format, which is processed via operating system emulation, the performance of the product is the processing and rendering of word processing file format data into a performance which a (human) user can experience via reading it off a display. The user can then interact with the processing (via for example entering, reformatting or deleting text) to change the data performance. Thus the measure of adequacy of the software is the satisfaction of the performance to the user when it is used to process input data, and thus how well it preserves the significant properties of its input data, and also preserving a known change in the performance which results from user interaction with the processing.

Thus, the adequacy of different preservation approaches is dependent upon the performance of the resulting replay on *data.* As the software has to be able to produce an adequate performance for any valid input date, the adequacy can be established by performing trial executions against representative test data covering the range of required behaviour (including error conditions). Additionally, the adequacy of preservation of a particular property can be established by testing against pre-specified suites of test cases with the expected behaviour, and pre-specified user interactions to change the data performance in known ways.

## The Conceptual Framework

In order to express the properties of software that need to be preserved for its effective long-term preservation, we have developed a conceptual framework to capture the approach taken to software preservation and the structuring of the software artefact and the significant properties of software for preservation.

### *A Conceptual Model for Software*

Various approaches to digital preservation have been proposed and implemented, usually as applied to data and documents. While these approaches to preservation vary in terms of implementation and other related technical aspects, they share in common an attempt to identify the additional information (i.e. metadata) needed to aid the preservation process. Examples include Functional Requirements for Bibliographic Records (FRBR) (International Federation of Library Associations (IFLA) Study Group, 1998), Preservation Metadata: Implementation Strategies (PREMIS) (PREMIS Working Group, 2005), and OAIS (CCSDS, 2002). We recognise that a conceptual

data model is required to determine the level of granularity at which preservation properties of software can be identified, and provide an understanding of the relationship between digital objects, thus giving traction on handling the complexity of the objects, a particularly important aspect in handling software. Therefore, we have developed a general model for software digital objects that is intended to provide a comprehensive view of the underlying dependencies of software, and thus help identify its preservation properties.

We define a general model for software consisting of four major conceptual entities in analogy with the FRBR model, which together describe a complete ***Software System***. These are *Product, Version, Variant* and *Instance* (Figure 3).

**Product:** The product is the whole top-level entity of the system, and is how the system may be commonly or informally referred to. Products can vary in size and can range from a single library function (e.g. a function in the NAG library[4]), to a very large system with multiple sub-products with independent provenances (e.g. Linux).

**Version:** A version of a software product is an expression of the product which provides a single coherent presentation of the product with a well defined functionality and behaviour. Note also that in composite products, the sub-products will themselves have a number of versions which will be related to versions of the complete product. These releases will not necessarily be synchronised, so the relationship between versions of sub-products need to be captured.

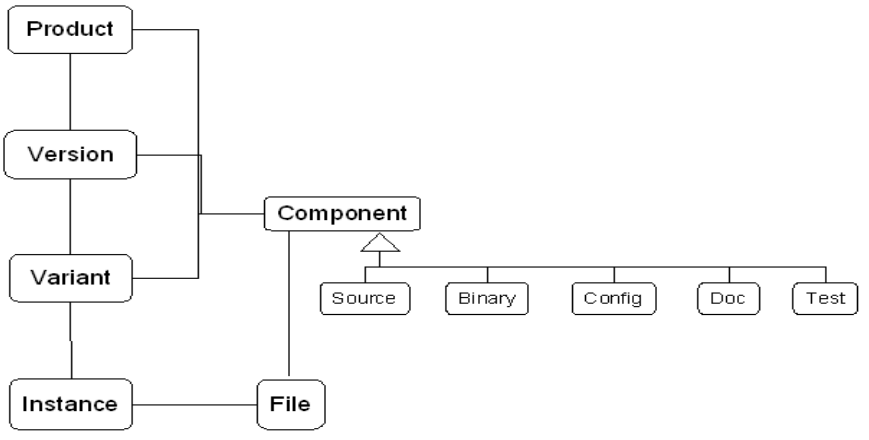


Figure 3. The Software Component Conceptual Model.

**Variant:** Versions may have a number of different variations to accommodate different operating environments, thus we define a *Variant* of the product to be a manifestation of the system which changes in the *software operating environment*, for example target hardware platform, target operating system, library, and programming language version. In this case, the functionality of the version is maintained as much as is practical; however, due to different behaviour supported by different platforms, there may be variations in behaviour, in error conditions and user interaction (e.g. the look and feel of a graphical user interface).

[4] Numerical Algorithms Group: http://www.nag.co.uk/

In practice, Version and Variant may be difficult to distinguish: changes in environment are likely to change the functionality; new versions of software are brought out to cope with new environments. It may be arguable that in some circumstances Versions are subordinate to Variants, and in others we may wish to omit one of these stages such as software which is only ever targeted at one platform. But it is worth distinguishing the two levels, as it makes a distinction between adaptations of the system largely to accommodate change in functional properties (versions), with those which are largely to accommodate change in properties of the operating environment (variants).

**Instance:** An actual physical instance of a software product which is to be found on a particular machine is known as an Instance. It may be also be referred to as an installation, although there is no necessity for the product to be installed; a master copy stored at a repository under a source-code management system may well not be executable within its own environment.

***Software Components***

All of the entities in the above conceptual model of software which form a software system are *composite*. Some of them may be subsystems, with sub-products. All systems however, will be constructed out of many individual *components*. A component is a storable unit of software which when aggregated and processed appropriately, forms the software system as a whole. Logical components typically (but not necessarily always) roughly correspond with a physical *file* (a unit of storage within an operating system's memory management). However, multiple components can be stored within in one file (e.g. a number of subroutines within one file) or across a number of files (e.g. help system or tutorial stored within a number of HTML files). Components may also be formed of a number of different digital objects, (e.g. text files, diagrams, sample data) which themselves would have preservation properties associated with their data format. A comprehensive preservation strategy for the full software system would have to consider those digital objects as well.

In this model, we give a number of different kinds of software component associated with a product, version or variant in the conceptual model of software in Figure 3. Note that this list is not exhaustive, and additional kinds of component may be identified. Of particular note is "Test Suite", which represents examples of operation of the product and expected behaviour arising from operation of the product. A test suite is typically produced to test the conformance of the product to expected behaviour in a particular installation environment. Thus, a test suite of a software product would play a significant role in measuring the adequacy of its preservation.

## The OAIS Reference Model and Software Preservation

The Reference Model for an Open Archival Information System (OAIS) is an ISO standard that is primarily concerned with the long-term preservation of digitally encoded information. In essence, the underlying notions of the OAIS reference model should be applicable to the long-term preservation of software artefacts as fundamentally (i.e. at bit level) they are in fact digitally encoded information.

Therefore, as illustrated in Figure 4, the OAIS information model can be applied to the process of rendering a preserved Data Source on a future technological platform, where the rendering of the data requires the use of a particular software product, which in turn requires a specific complier, to be rebuilt from its preserved state. In short, the OAIS defined Descriptive Info, Representation Information (RI) and Preservation Description Information (PDI) (CCSDS, 2002) can be used to retrieve (discover and access), reconstruct (compile source code), and replay (verify authenticity and run) a software object respectively.

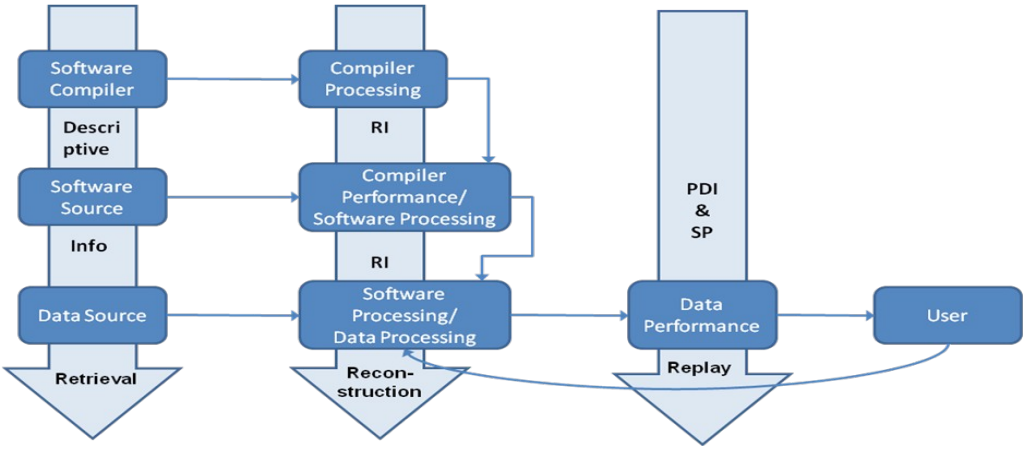


Figure 4. The Relationship between the OAIS Information Model and the Software Performance Model.

However, once re-built, **Significant Properties (SPs)** about the software are required to measure the adequacy of the software in processing the Data Source, which in turn measures the performance of the compiler in re-building the software from its source code. This is not comprehensively addressed in the OAIS model but may be considered amongst the **Preservation Description Information** of software for demonstrating the satisfaction of significant properties, and thus viewed as an additional component of the OAIS information object in the context of long-term software preservation.

## Preservation Properties of Software

In considering what preservation properties are needed for software, we need to consider the following seven general categories of features which characterise software (Table 1). These categories apply to each of the four major conceptual entities of a software system defined in the conceptual model of software .We also try to demonstrate the relationship of each of these categories to the relevant OAIS information entity.

| Category | Description | Examples | Equivalent OAIS Terms |
|---|---|---|---|
| **Functionality** | • Description of the typical characteristics of software.<br>• Useful for efficient discovery and accessibility of the software in future | • Description of inputs and outputs<br>• Description of operation and algorithms<br>• Description of the domain addressed | • *Descriptive Information* |

| | | | |
|---|---|---|---|
| **Software Composition** | • Description of the components that constitute software<br>• Useful for rebuilding and reusing the software in future<br>• Detailed history of version changes and other significant changes that a software product has undergone facilitates verification of its authenticity in future. | • A typical record: binary files, source code, user manuals and tutorials.<br>• A more complete record: requirements and design documentation, test cases and harnesses, prototypes, formal proofs. | • *Representation Information*<br>• *Preservation Description Information (PDI)* |
| **Provenance and Ownership** | • Different software components have different and complex licensing conditions.<br>• Needs to be included in the preservation planning | • Software owner and licence information, e.g. Microsoft for MS Word® | • *Provenance Information* category of *Preservation Description Information (PDI)* |
| **User Interaction** | • Description of expected mode of interaction between user and software<br>• The 'Look and Feel' and the model of user interaction can play a significant role in the usability of the software and therefore should be considered among its ***Significant Properties***. | • The inputs which a user enters through a keyboard, pointing device or other input devices, such as web cameras or speech devices<br>• The outputs to screens, plotters, sound processors or other output devices | • *Not comprehensively addressed in the OAIS – may be categorized as the Significant Properties of software* |
| **Software Environment** | • Description of the environment that the correct operation of the software depends on<br>• Dependencies between software environment related entities and history of changes made to them | • Hardware platform, operating system, programming languages and compilers, software libraries, other software products, and access to peripherals.<br>• Binaries usually require an exact match of the environment to function | • *Representation Information* |
| **Software Architecture** | • Plays a significant part in the reproducibility of the original functionality and features of software | • Client/server, peer-to-peer, and Grid systems all require different forms of distributed system interaction which would require the configuration of hardware and software to be reproduced to reproduce the correct behaviour. | • *Representation Information* |

| | | | |
|---|---|---|---|
| **Operating Performance** | • The performance of the software with respect to its use of resources (as opposed to its performance in replaying its content)<br>• Plays a significant part in the reproducible behaviour of software.<br>• Contributes towards the information needed to measure the overall adequacy of software preservation in future | • Speed of execution, data storage requirements.<br>• In some circumstances, we may wish to replay the software at the original operating performance rather than a later improved performance.<br>• A notable example of this is games software, which if reproduced at a modern processor's speed would be too fast for a human user to play. | • *Not comprehensively addressed in the OAIS – may be categorized as the Significant Properties of software* |

Table 1. Different categories of preservation properties of software.

## Applying the Software Preservation Model to the BADC Software

We carried out a series of case studies into existing practices of software preservation and maintenance in order to validate the applicability of the software preservation model. Of particular note among these studies is the one conducted on the British Atmospheric Data Centre (BADC)[5], which involved evaluating the model against a number of BADC software. For this, we tried to collect the appropriate value(s) for each of the preservation properties defined in the framework for each major conceptual entity of software. Table 2 outlines the preservation properties of "Product" entity of the BADC Web Feature Service[6] identified as part of the BADC case study.

[5] The British Atmospheric Data Centre: http://badc.nerc.ac.uk/home/index.html

[6] The BADC WFS Enables retrieving and updating geospatial data encoded in Geographic Markup Language (GML): http://www.opengeospatial.org/standards/gml), or any GML-based formats, irrespective of the location or storage media of the data. The implementation is based on the Open Geospatial Community (OGC) standard for Web Feature Service: http://www.opengeospatial.org/standards/wfs

| Property Category | Software Property | |
|---|---|---|
| | **Name** | **Value** |
| **Functionality** | **Purpose** | Enabling publishing and querying of Geospatial data on the web using open standards |
| | **Keyword** | Web feature service |
| **Provenance and Ownership** | **package_name** | GeoServer |
| | **Owner** | GeoServer and SeeGrid |
| | **Licence** | GNU GENERAL PUBLIC LICENSE, Version 2<br>http://geoserver.org/display/GEOS/License |
| | **Location** | http://geoserver.org/display/GEOS/Welcome |
| **Software Architecture** | **Overview** | The software architecture is comprised of a series of modules for handling requests for geospatial data as geographical features across the web using platform-independent calls, such as HTTP Get and Post and SOAP. http://geoserver.org/display/GEOSDOC/1+GeoServer+Architecture |
| **Software Composition** | **Software overview** | http://geoserver.org/display/GEOS/What+is+Geoserver |
| | **Tutorials** | Installation: http://geoserver.org/display/GEOSDOC/1+Getting+Started |
| | **Requirements** | Operating system: Window/Linux/Unix, Minimum RAM: 512 megabyte, Java 1.5 or higher |

Table 2. "Product" properties of BADC GeoServer/WFS.

The experience of applying the framework for software preservation to the BADC WFS/GeoServer software shows that the framework is sufficiently relevant to the software (e.g. GeoServer) used as well as being adequate in terms of the information recorded. However, it also highlights the necessity to have considerable knowledge of both the framework and software in question to accurately apply the framework to the software. This indicates a need for tools to facilitate the recording of software preservation properties by providing guidelines which, for example, explain the underlying concepts of the framework in a user-friendly manner.

## SPEQS

In the light of the findings from the BADC case study, we have developed a tool, called **Significant Properties Editing and Querying for Software (SPEQS)** to support the systematic collection of preservation properties for software. In essence, SPEQS exemplifies how the capture of the preservation properties identified in the software preservation framework could be integrated within the software development lifecycle. It has been implemented in Java as a plug-in for Eclipse[7], a widely used open source interactive software development environment, to enable software developers to record, edit and query preservation properties of software (as defined in Table 1) directly from within the Eclipse environment. It also provides software developers

[7] Eclipse: http://www.eclipse.org/

with suitable guidelines for accurately recording significant properties of software. We envisage that this approach of enabling capturing preservation properties of software during its development lifecycle would contribute towards ensuring the accuracy of the information recorded.

### *Architecture Overview of SPEQS*

SPEQS uses an ontology representation of the conceptual model for software, written in OWL (Web Ontology Language)[8] for recording preservation properties of software (SPs) in RDF[9] format and querying the recorded SPs using SPARQL[10], the query language for RDF. The SPEQS architecture consists of four principal components: ***the SP Editor, the SP Query Interface, the SPEQS Data Store*** and ***a software repository***, such as Subversion[11] (Figure 5).

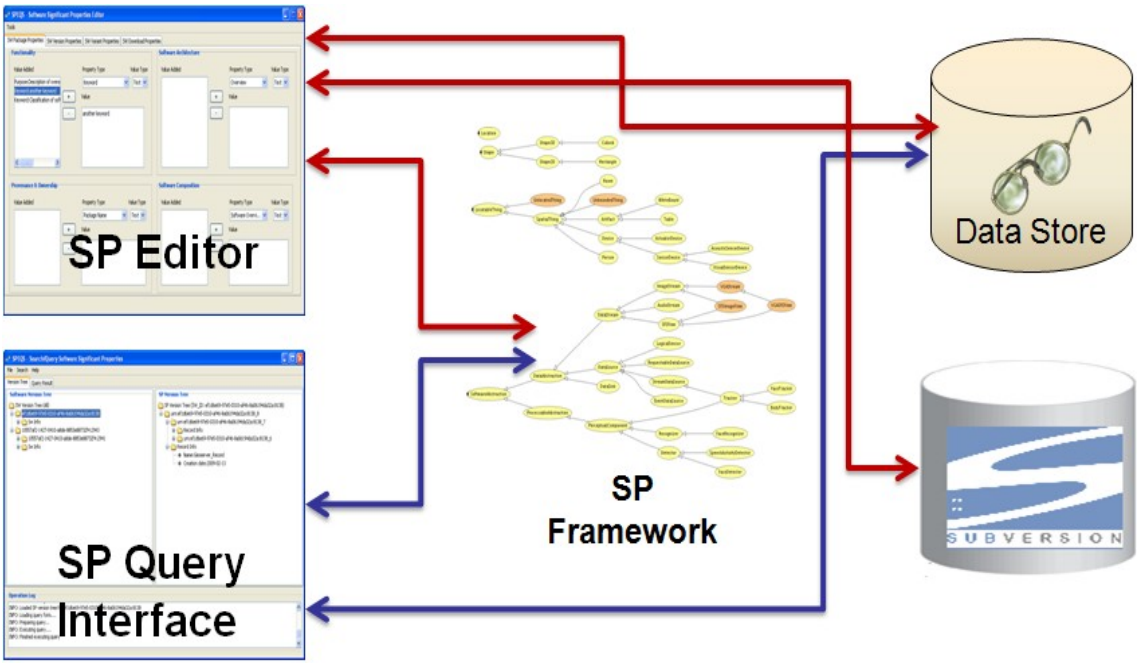


Figure 5. An Architectural View of SPEQS.

The SP Editor and the SP Query Interface are Graphical User Interfaces (GUIs) implemented using the Java Swing API[12] and accessible from the SPEQS menu of Eclipse toolbar. The SP Editor uses the Jena[13] OWL/RDF API to enable recording and updating of SPs for software projects defined within an Eclipse environment. And the SP Query Interface enables querying, viewing and analysing the recorded SPs using the Jena SPARQL query engine.

The SPEQS Data Store is a relational database that consists of a RDF Triple Store[14] for storing SP records and a standard data storage entity for storing other meta-information (e.g. developer name, creation date etc.) associated with a software project. In addition, SPEQS interacts with software repositories and management

[8] OWL: http://www.w3.org/TR/owl-features/
[9] Resource Description Framework (RDF): http://www.w3.org/RDF/
[10] SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/
[11] Subversion: http://subversion.apache.org/
[12] Java Swing: http://java.sun.com/j2se/1.5.0/docs/guide/swing/index.html
[13] Jena – A Semantic Web Framework for Java: http://jena.sourceforge.net/
[14] Databases specially configured to store and enable querying large RDF models.

systems, such as Subversion, to keep track of the changes made to software and ensuring accurate and consistent association with its preservation properties. At present, SPEQS only supports Subversion based software projects.

***Evaluation of SPEQS***

SPEQS was evaluated against a number of software products. In particular, it was effectively used in the CASPAR (Cultural Artistic and Scientific knowledge for Preservation, Access and Retrieval)[15] project for capturing preservation properties of SAO Explorer software that enables visualising and analysing Ionosonde data. The preservation properties of SAO Explorer captured using SPEQS contributed towards modelling a comprehensive preservation network for the Ionosonde data (Conway, Dunckley, McIlwrath, and Giaretta, 2009).

However, there is still considerable scope for further improvement in SPEQS. In particular, SPEQS needs to incorporate an efficient mechanism for semantically validating values asserted in a SP record. This could involve integrating SPEQS with a suitable controlled vocabulary. Furthermore, the SPEQS Data Store should be subjected to effective long-term preservation technique, e.g. by integrating it with an efficient long-term preservation archive, to ensure longevity of the SP records. Additionally, to cater for a wider range of software projects, SPEQS would benefit from incorporating support for other widely used software development environments, such as NetBeans[16] and other software repositories, such as SourceForge[17] and CVS[18]. Despite these shortcomings, we believe that SPEQS demonstrates the potential of a comprehensive software system that would facilitate capturing, validating, and querying preservation properties of software.

## Conclusions

In this report we have developed a conceptual framework to express a rigorous approach to software preservation. It develops and extends the notion of performance and emphasises the notion of adequacy and relates it to authenticity; proposes a measurement of adequacy to the preservation of properties which are testable within a performance; and uses the concepts introduced within the OAIS model within the framework to categorise the identified preservation properties. Thus this framework can be seen as a specialisation of the OAIS model to handle the case of software.

We believe that this is a general and principled approach which can cover the preservation needs of a wide range of different software products, including modern distributed systems and service oriented architectures, which are typically built of pre-existing frameworks and have a large number of dependencies on a widely distributed network of services, many of which are outside the control of the typical user (e.g. DNS services, proxies, web services provided by external organisations such as Amazon Web Services[19]). Further, the performance model presented here, which has a notion of user feedback to influence the performance, may represent an approach to preserving the user interface and the user interaction model, although work is required to further develop that notion.

[15] CASPAR Project: http://www.casparpreserves.eu
[16] NetBeans: http://www.netbeans.org/
[17] SourceForge: http://sourceforge.net/
[18] Concurrent Versions System (CVS): http://www.nongnu.org/cvs/
[19] Amazon Web Services: http://aws.amazon.com/

Further work is required to test this model and to provide tools. Some initial work has been undertaken to integrate the capture of preservation properties of software within a software development process, and also to use the framework within case studies. Further work on case studies is required, especially across a range of software types to cover the diversity of software and to consider how to support the preservation of legacy software.

## Acknowledgements

We would like to thank our colleagues David Giaretta, Esther Conway, Steven Rankin, Brian McIlwrath and other members of the Digital Curation Centre and CASPAR projects for their advice and discussions, and to Jim Woodcock of the University of York for contributing to case studies. The work was carried out under the JISC study into the Significant Properties of Software and the JISC project Tools and Guidelines for Preserving and Accessing Software Research Outputs.

## References

Consultative Committee for Space Data Systems (CCSDS). (2002, January). *Reference Model for an Open Archival Information System (OAIS)*. Recommendation for Space Data System Standards. CCSDS Blue Book. Washington, D.C.:CCSDS Secretariat. Retrieved July 27, 2009, from http://public.ccsds.org/publications/archive/650x0b1.pdf

Conway, E., Dunckley, D., McIlwrath, B., & Giaretta D. (2009, December). Preservation network models: Creating stable networks of information to ensure the long term use of scientific data. *Ensuring Long-Term Preservation and Adding Value to Scientific and Technical Data (PV 2009). Villafranca del Castillo, Madrid, Spain.* Retrieved November 13, 2009, from http://epubs.cclrc.ac.uk/bitstream/4314/PV09_Conway_PNM.pdf

Heslop, H., Davis, S., & Wilson, A. (2002). *An Approach to the Preservation of Digital Records*. National Archives of Australia. Retrieved July 29, 2008, from http://www.naa.gov.au/Images/An-approach-Green-Paper_tcm2-888.pdf

International Federation of Library Associations (IFLA) Study Group. (1998). *Functional Requirements for Bibliographic Records.* Retrieved July 27, 2009, from http://www.ifla.org/VII/s13/frbr/frbr.pdf

Matthews, B.M., McIlwrath, B., Giaretta, D., & Conway, E. (2008). *The Significant Properties of Software: A Study.* In JISC report, 2008. Retrieved August 3, 2009, from http://sigsoft.dcc.rl.ac.uk/twiki/pub/Main/SigSoftTalks/SignificantPropertiesofSoftware.doc

Matthews, B.M., Shaon, A., Bicarregui, J.C., Jones, C.M , Woodcock, J.C.P., Conway, E. (2009a). *Towards a methodology for software preservation* iPres 2009 The Sixth International Conference on Preservation of Digital Objects. October 2009.

Matthews, B.M., Shaon, A., Bicarregui, J.C., Jones, C.M , Woodcock J.C.P. (2009b, December). An approach to software preservation. *PV 2009 Ensuring Long-Term Preservation and Adding Value to Scientific and Technical Data.*

PREMIS Working Group. (2005, May). *Data Dictionary for Preservation Metadata.* Retrieved July 27, 2009, from http://www.oclc.org/research/projects/pmwg/premis-final.pdf