A Distributed Archival Network for Process-Oriented Autonomic Long-Term Digital Preservation

Inauguraldissertation

zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Ivan Subotic
aus Basel (Basel-Stadt)

Basel, 2013

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät

auf Antrag von

Prof. Dr. Heiko Schuldt, Dissertationsleiter
Prof. Dr. Andreas Rauber, Korreferent

Basel, den 26. Februar 2013

Prof. Dr. Jörg Schibler, Dekan

Abstract

The rapidly growing production of digital data, together with their increasing importance and essential demands for their longevity, urgently require systems that provide reliable long-term preservation of digital objects.

These systems have to ensure guaranteed availability, integrity, authenticity, and interpretability over the course of the preservation, where the preservation period may last for several years, for instance in business or scientific applications, the lifetime of a human in medical applications, up to potentially unlimited time-spans for preservation in cultural heritage digital libraries. This means that all kinds of technical problems (network, software or hardware failures) need to be reliably handled and that the evolution of data formats is supported. At the same time, systems need to scale with the volume of data to be archived. Thus, long-term digital preservation systems have to be inherently distributed to allow content to be replicated. Institutions with long-term archiving needs for the preservation of digital data, have to collaborate in order to build a highly reliable and available, geographically distributed Internet-based digital archiving system. By employing distributed systems technologies be it for the creation of a small cooperating network of few institutions with limited resources, or a large network with many nodes providing combined potentially vast amounts of globally distributed resources, the challenges lie in the autonomic, efficient, and fault-tolerant use of these resources without a centralized global coordinator.

We present novel concepts for a distributed long-term preservation system for digital data, with a focus on long-term preservation as required by archives, museums, research communities, or the corporate sector. These concepts are the result of combining distributed, autonomic, and process oriented computing, with requirements from the digital preservation community regarding special system, user, and metadata functionality. Originating from this fusion, our novel concepts are the main ingredients of the described system model, consisting of a data model, and different processes. At the data level, support is provided for complex data objects, management of collections, annotations, and arbitrary links between digital objects. At process level, our proposed archiving system model supports automated processes that provide dynamic replication, consistency checks, and automated recovery of the archived digital objects utilizing autonomic behavior governed by preservation policies without any centralized coordinator in a fully distributed network. This allows for an efficient and fault-tolerant use of the resources provided in the network.

Further, we present a prototype implementation of the DISTARNET (DISTributed ARchival NETwork) system, a distributed long-term digital preservation solution, which utilizes the described novel concepts. While implementing the described data model and processes, our implementation is additionally governed by considerations such as fault-tolerance on the node level, maintainability and extendability, and long-term use of the system, which all flow into the described system architecture, and resulting implementation.

Subsequently, we then perform an evaluation of the implemented prototype and the underlying concepts, with the use of realistic scenarios. The evaluation consists of two parts. In the first part, we define and employ a benchmark geared towards triple stores, in which we evaluate the feasibility and the constraints of using triple stores for RDF-based metadata storage and management in the context of long-term preservation systems. In the second part, we perform a qualitative and quantitative evaluation of the DISTARNET system prototype implementation. The former looking at the correct execution of the developed processes, and the later looking at the performance of the system regarding the overall archiving storage capacity and scalability of the system.

Acknowledgements

I would like to express the deepest appreciation to my thesis advisor and mentor Prof. Dr. Heiko Schuldt for his extraordinary supervision, and for the generous and friendly support he provided over these years while displaying a lot of patience. With his stimulating discussions he gave me a lot of insight which I have been able to express in this thesis. Also he provided me the huge opportunity to write this thesis in his group, and to say the least, this thesis would not have been possible without him.

I wish to thank Prof. Dr. Andreas Rauber from the Vienna University of Technology in Austria, for kindly agreeing to be my co-referee.

I am particularly grateful for patient guidance, the support, and advice given by P.D. Dr. Lukas Rosenthaler, Prof. Dr. Rudolf Gschwind, Dr. Simon Margulies, and all the great people at the Imaging and Media Lab at the University of Basel.

I would also like to thank Daniela Bienz, for her loving support and enthusiastic encouragement during this time. Finally, I wish to thank my father, Branislav Subotic, for his patience and continuously support throughout my thesis.

M∴ B∴

Contents

List of Figures

List of Tables

1 Introduction

Digital data, either digitized or digital born are increasingly gaining importance in our everyday life. As a consequence, a large spectrum of applications require that data is preserved over long periods of time — up to several years due to legal constraints in business applications or for scientific data, for the duration of the lifetime of a human in medical applications, up to potentially unlimited time spans for the preservation of cultural heritage. Approaches to digital long-term preservation are constrained by the enormous and ever growing volumes of data. In addition, long-term data archiving and preservation also needs to take into account that data has to outlive the hardware on which they are stored and the data formats in which they are represented.

It is essential that at access time in the future as a result of the digital preservation task, the information carried by the initially stored digital object, is still accessible and usable. Metadata is the key to providing long-term digital preservation. The Open Archival Information System (OAIS) Reference Model [1, 2] categorizes metadata that need to be preserved and managed together with the original bitstream into the following categories: representation information (to allow the data to be rendered and used as information); reference information (to identify and describe the content); context information (for example, to document the purpose for the creation of digital content); fixity information (to permit checks on the integrity of the digital content); and provenance information (to document the chain of custody and any changes since the content was originally created). There is also additional metadata that is created during the preservation process when a digital object evolves (e.g., annotations, links between information objects, collection/subcollection information).

We will use the term *Information Object* to denote the digital data (bitstream), their metadata and optional links to other objects.

The long-term preservation of digital information objects is challenged by a number of problems and risks (e.g., network, hardware and software failure, data format obsolescence, etc.) that need to be addressed for a successful preservation outcome. In order to escape from the technological obsolescence of specific hardware and software products, the OAIS reference model explicitly considers the migration of digital data to new data carriers and/or data formats in periodic intervals

In short, digital long-term preservation combines policies, strategies, and actions for preserving information objects, despite potential changes of the formats in which objects are stored, and in the underlying hardware environment. Therefore, a software system for digital long-term preservation has to support preservation processes that guarantee (i) *integrity*: the information captured by data is not altered in any way; (ii) *authenticity*: provenance information is properly linked to each stored object by means of appropriate metadata; (iii) *chain of custody*: location and management controls are tracked within the preservation environment; (iv) *completeness*: everything that was initially stored is also available in the future and finally (v) *ease of access*: the necessary means are provided to find and identify the stored digital objects. Moreover, an essential requirement for viable long-term preservation systems is their capability to do the necessary maintenance and failure recovery in an *autonomous* way, e.g., to automatically identify when a pre-defined replication level is no longer reached and to trigger corrective actions (deploy new replicas) without human intervention.

To cope with all these challenges, we have developed novel concepts for a distributed long-term preservation system for digital data, with a focus on long-term preservation as required by archives, museums, research communities, or the corporate sector. These concepts are the result of combining distributed, autonomic, and process oriented computing, with requirements from the digital preservation community regarding special system, user, and metadata functionality. The goal of the development is to provide concepts, when implemented in a software system, the deploying institutions can, on their own or through collaboration with others, build an autonomous, reliable replicated, geographically distributed, Internet-based digital archiving system.

The concepts describe a data model, and a number of processes. At the data level, support is provided for complex data objects, management of collections, annotations, and arbitrary links between digital objects. At process level, we provide support for automated processes. The processes provide dynamic replication, consistency checks, and automated recovery of the archived digital objects utilizing autonomic behavior governed by preservation policies without any centralized coordinator in a fully distributed network.

Further, we present a prototype implementation of the DISTARNET (**DIST**ributed **AR**chival **NET**work) system, a distributed long-term digital preservation solution, which utilizes the described novel concepts, and additional considerations regarding fault-tolerance, maintainability and extendability, and long-term use.

1.1 Scenarios

In the following, we briefly sketch three use case scenarios that highlight the broad applicability of DISTARNET, as a viable solution for a reliable, cost effective, and efficient long-term archiving system. DISTARNET stands in these scenarios as one possible representative for systems implementing the developed concepts.

1.1.1 Multinational Pharmaceutical Corporation

Carol, head of the computer science department at ACME Ltd. a large multinational pharmaceutical corporation, has the task to implement a new digital archiving solution which is compliant to the company's preservation policy. In addition to the standard requirement for digital preservation (integrity, authenticity, etc.), this policy imposes that data has to be redundantly stored at least three different locations, with added constraints regarding minimum distance between locations and – for some types of data – also the country or state in which the data is allowed to be stored. Other issues like enforcing integrity (uncorrupted record), authenticity (linking of provenance information to each record), chain of custody (tracking of location and management controls within the preservation environment, e.g., who, where, and when handled the archived data in the network), trustworthiness (sustainability of the records), and readability (long-term access through data format migration) need also to be addressed. ACME Ltd. operates several data-centers worldwide, which Carol can use for deploying her new archiving solution.

Carol and her team decide to use DISTARNET for her archiving tasks. She deploys DISTARNET nodes at several data centers of ACME Ltd. and specifies additional policies. These nodes connect to each other and exchange information on the available storage resources. As soon as data gets ingested at either of the nodes, DISTARNET will autonomously search for suitable nodes inside ACME's DISTARNET network and initiates the creation of replicas. In normal mode, DISTARNET periodically checks the integrity of the archived data and synchronizes changes (e.g., propagates new annotations that have been made). All this is handled automatically, without the need of manual intervention. In case of failures (e.g., nodes become unavailable or data are corrupted due to hardware and/or software failure), the built-in monitoring functionality of DISTARNET will detect the problem and will automatically initiate countermeasures by instantiating workflow processes that create additional replicas at other suitable nodes adhering to the specified policies.

Once the problem is solved, DISTARNET will automatically pull the digital data from the DISTARNET network onto the repaired node and deletes any excess replicas from the least suitable nodes. Examples of organizations that comply to this scenario include memory institutions such as large libraries, foundations, film archives etc., which are dedicated to preserving cultural heritage. Other examples include large pharmaceutical companies which have to preserve their research results due to legal constraints.

1.1.2 National Museum of History & Native Art

Jim, a digital archivist at the National Museum of History & Native Art in a small European country, wants to implement a new archiving solution for preserving his country's cultural heritage. This new solution should enable him to have redundant off-site replicas, although the museum itself has only one site available that can be used to deploy such a solution. However, there is a collaboration agreement between the national museums of different countries that include access to the other institutions' computation and storage resources for deploying replicas, together with the enforcement of access restrictions on these shared data (pretty much like in a virtual organization known in the context of grid computing). As before the issues of data integrity, authenticity, chain of custody, and trustworthiness need to be addressed.

Each museum deploys a DISTARNET node. When data is ingested, they will be handled according to the policies specified by their owner and will automatically be distributed across the storage resources of different museums. Again as before, DISTARNET will also periodically instantiate maintenance processes and launch recovery processes when necessary, such as periodic integrity checks of the archived digital data and autonomously initiated replication, that will ensure adherence to the preservation policies. In case of error, e.g., corrupted or lost replicas, etc., the system will autonomously initiate countermeasures without the need of external intervention.

Fast forwarding into the not so far future, the system will send reminders to Jim, based on data format migration policies defined in the past, about the possible need to migrate the archived content to newer formats. He will be able to initiate data format migration processes, that will migrate the older formats to the new defined formats in the background as needed.

This scenario represents institutions in the cultural heritage community. We will just mention two typical examples of organization with whom the author has collaborated in the past. First, municipal libraries like the Denver Public Library (DPL). In this case, the digital collection consists of about 120,000 digitized images of its Western History Collection[1]. The master images are stored on CD-Rs (two copies of each CD). In a publicly available paper[2], the DPL writes:

> "The Library's initial plan was to transfer the archive files to the best available storage media after 10 years. As we approach this date, we have discovered that the issue may be more complicated. Transfer of the complete collection of archive files will be time consuming and costly."

Second, the e-codices project[3], funded by the Mellon Foundation, which is establishing a virtual library of most of the medieval handwriting and codices in Switzerland. Currently, the digital data is "archived" in two copies on consumer-level hard-disks sitting on a shelf in the project office.

1.1.3 Cloud Storage Provider

The cloud storage provider Stratocumulus Inc. plans to release DA^{3}S (Data Archiving as a Service), a new data management service. Essentially, DA^{3}S offers customers the

[1]`http://digital.denverlibrary.org`
[2]`http://history.denverlibrary.org/images/photodigitization_project.pdf`, p.40
[3]`http://www.e-codices.unifr.ch/`

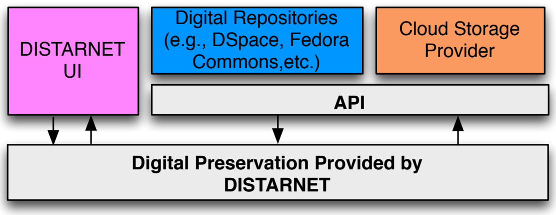
DISTARNET UI
Digital Repositories (e.g., DSpace, Fedora Commons,etc.)
Cloud Storage Provider
API
Digital Preservation Provided by DISTARNET

Figure 1.1: DISTARNET Deployment Categories

option for long-term digital preservation of their digital assets, with dedicated quality of service guarantees on data availability (replication), integrity (regular checks), and authenticity. For this, Stratocumulus Inc. installs DISTARNET on each of their data-centers. In this setting, DISTARNET will be a layer underneath the cloud so that Stratocumulus Inc. can provide the services offered by DISTARNET, fully transparent to their end users. As optional services for DA^{3}S for an extra cost, Stratocumulus Inc. offers automated data format migration. Moreover, storage location constraints can be defined through preservation policies that will allow to restrict, where data will be stored since Stratocumulus Inc. runs multiple data-centers around the world.

1.1.4 Deployment Categories

As we have seen in the three scenarios, DISTARNET can take a more visible role where the user has direct contact with the system (Scenarios 1 + 2), or a role in the background, where the deploying party is providing services, enabled by DISTARNET (Scenario 3). Figure 1.1 shows the positioning of DISTARNET in regard to other system categories.

In all three mentioned scenarios DISTARNET can offer a viable solution (or be part of the solution) for a reliable, cost effective and efficient long-term archiving system. DISTARNET is able to provide a high degree of scalability, by relying on concepts from Peer-to-Peer systems and Grid computing domains for managing distributed storage resources. The scalability provided is both in terms of volumes of digital content to be archived and in terms of the available storage resources.

1.2 Challenges of Digital Long-Term Preservation

The challenges that *Digital Long-Term Preservation Systems* are faced with are comprised of general issues distributed systems need to handle (e.g., fault tolerance, scalability, load balancing, security) and additional issues arising from the specific requirements for digital long-term preservation (e.g., integrity of complex information objects, authenticity, data format obsolescence, long-term readability, ease of access). In the following, we will discuss these challenges.

Distribution. To provide a secure environment for long-term storage of the archived object, we need to store more then one copy of each object. Also, they should ideally not be stored in one place. Thus, we need some form of a distributed and replicated storage environment.

Fault Tolerance and Failure Management. The failure of one or more components in a distributed preservation system should not endanger the whole system, and should only have isolated effects. Failure (e.g., power failure, hardware failure, etc.) or disaster (e.g., natural disaster, fire, etc.) resulting in the destruction or corruption of some of the stored information objects should not result in a complete loss of the archived data. Automated replication mechanisms should maintain a minimum number of geographically dispersed replicas (number and location defined by preservation policies) of the stored information objects. Any data loss event should trigger automated recovery processes that will reestablish the minimum number of geographically dispersed replicas. This should be done by either using the repaired failed storage nodes, or by using other available and suitable storage nodes found through resource discovery.

Management of Complex Information Objects. The long-term preservation of digital data requires the management of complex information objects, i.e., information objects that are comprised of or are part of other information objects. The complexity arises out of the requirement for preserving additional supporting information beside the bitstream of the archived data object (representation, reference, context, fixity, provenance, and other information).

The challenge lies in the automated management of such complex objects in a distributed setting. Preserving the integrity of complex objects is a twofold problem. First, the integrity of the referential information needs to be maintained (e.g., are all references properly defined and are all objects referred to available), and second, the integrity of the objects themselves. Referential and object integrity checking (through the use of fixity information stored with the complex object) needs to be automated. Any loss of integrity needs to trigger automated processes that will restore the integrity of the information object. If the information object cannot be repaired solely by the information it carries itself, other remote replicas need to be used. As an example, through hardware failures some information objects might be corrupted or destroyed. The information object representing a collection is partially corrupted while some of the objects that are part of the collection are destroyed. In this case, the discovery and subsequent recovery of the referential information (and through inference also of the endpoint information objects) need to be automated. Integrity is also an important challenge in the context of synchronization of information objects that are changing during the preservation process (e.g., annotations, links between information objects, collection/subcollection information). A system needs to make sure that such changes do not break (falsely) the integrity of the information objects.

Scalability. The growing production of digital data that need to be archived requires a scalable distributed preservation system. A system that should work efficiently even with an increasing number of users and quickly growing volumes of data that needs to be stored. The addition of storage resources should enhance the performance of the system. This requires that the processes supporting the archiving operations are automated and scalable themselves.

Openness and Extensibility. A long-term preservation system should provide clearly separated and publicly available interfaces to enable easy extensions to existing components and the possibility of adding new components. The system should be able to be adapted to arising new challenges, by allowing curators of digital objects to specify new processes to cope with additional challenges (e.g., novel data formats).

Resource Discovery and Load Balancing. In a distributed preservation system, the discovery of newly available resources, together with the monitoring and management of existing resources is very important and should be handled efficiently. The information gathered is important for the functioning of processes that provide automated replication of the information objects to suitable remote storage nodes (constrained through preservation policies). The system should be able to distribute the replicas among the available resources for improving performance, where performance incorporates measures such as availability, access speed, higher security, and reliability. Dynamically incorporating new resources or correctly handling the loss of existing resources (temporarily or permanently) should be provided via automated processes. For an efficient usage of the available resources, tasks (e.g., data format migration) should be executed immediately or deferred baring the availability of the resources needed.

Authentication, Authorization, and Auditing. Access to resources should be secured to ensure only known users are able to perform allowed operations. Apart from these usual security precautions, in a distributed preservation environment (e.g., different institutions cooperate and share storage space), only the institution should be able to access and manage its owned data. No access should be possible to foreign data hosted on the local node for redundancy reasons. Cooperating institutions should be able to access other institution's meta-data. Also, after having been authorized by the data owner be granted access to the content of interest . The system must further support error reporting, and provide logging.

1.3 Contribution and Scope of the Thesis

In the context of long-term digital preservation, the reliable and fault-tolerant management of large digital archives poses requirements which cannot be met by individual organizations but need the collaboration of different, geographically dispersed organizations to build novel Internet-based digital archiving systems.

In this document, we present novel concepts for a distributed long-term digital preservation system. We describe the challenges and risks that need to be addressed, and present our developed flexible and reliable approach to digital preservation, in particular the data model, and the predefined processes for maintenance and failure handling purposes.

The main contribution lies in the detailed analysis of the concepts providing self-$*$ capabilities, i.e., how a system implementing these concepts is able to automatically adjust itself to changing environments (both in terms of volumes of digital content to be archived and the available storage resources) and how it automatically recovers from different kinds of failures. The autonomic behavior also includes the compliance to quality of service guarantees for the users (digital archivists) such as a predefined level of data availability or specific constraints on the locality of data and replica placement.

Further, we present a prototype implementation of the DISTARNET (**DIST**ributed **AR**chival **NET**work) system, a distributed long-term digital preservation solution, which utilizes the developed concepts. During the description of the implementation, we discuss additional considerations governing the implementation. These considerations include fault-tolerance on the node level, maintainability and extendability of the system components, and long-term use of the system, which all flow into the described system architecture, and resulting implementation.

Self-Configuration. The concepts provide the ability to automatically detect changes to the distributed archiving network. New nodes joining or nodes leaving are being constantly monitored, and their current status is taken into account. For example, periodic neighbor-nodes checks, where every node checks periodically his neighbors by sending a message to which the receiver has to reply in a defined time. If the receiver does not reply than the node is marked as lost after some defined period of time. After that, the system will start with self-healing.

They further provide automated dynamic replication of data. This includes replicating data in a safe manner and keeping the replicas in sync. The data replication task needs to be automated in such a way, so that the defined redundancy (via policies) is always upheld.

Self-Healing. The concepts provide the ability to recognize abnormal conditions or problems that may be harmful, and the ability to recover from them. For example in the case of a "Node-Lost Event", where countermeasures are automatically initiated (e.g., create additional replicas, etc.) needed to uphold the specified redundancy which was defined in the policies. Another example would be the periodic integrity (bitstream and referential integrity) checking that if breached would automatically trigger countermeasures (e.g., find healthy replicas and copy them in place of the broken one) to rectify the problem.

Self-Optimization. The concepts provide the ability to know the environment, surroundings, and other resources available, and automatically detect any changes concerning the participating nodes (capacity, location, uptime, connectivity, speed, etc.). This information will be used to autonomously manage and maintain resource allocation (e.g., finding suitable nodes where data can be replicated to, automatic policy-based geographical distribution of data, etc.).

1.4 Structure of the Thesis

This thesis is organized in six chapters. In Chapter 2 we begin by presenting the current state-of-the-art of digital preservation by talking about current employed standards and practices. This information is geared toward readers coming from the computer science community to provide a solid foundation of the preservations' community terminology and methods. The second part of this chapter is aimed at readers coming from the preservation community, to provide them with a solid foundation of the used computer science terminology, concepts, and methods. Chapter 3 presents general requirements for a long-term preservation system followed by a discussing of the developed DISTARNET concepts. Following, Chapter 4 discusses the implementation, and Chapter 5 the evaluation of the implementation in regard to the requirements. Chapter 6 presents related work, and we conclude in Chapter 7.

2

Digital Preservation and Distributed Systems Foundations

We begin this chapter by providing a general introduction to Digital Preservation and surrounding topics. This information is geared toward readers coming from the computer science community to provide them with a solid foundation of the terminology and methods used in the preservation community, and which are relevant to the understanding of the later chapters. Further, we also provide a general introduction to the used computer science terminology, concepts, and methods surrounding Distributed Systems, which is aimed at readers coming from the preservation community.

2.1 Digital Preservation

The growing production of digital data, be it born digital or retro-digitized artifacts (e.g., photographs of photographic collection, digitized movie film, sound, etc.) challenges archiving institutions with new problems and new needs for the secure long-term preservation. Long-term preservation in general is defined as the task to guarantee the authenticity, access, and the interpretability of archived assets in a usually not determined far future.

The preservation of digital data is different from the traditional preservation of analog sources and records in a way that it is not necessary to preserve physical evidence of the artifacts or records but rather the semantic content. In this aspect, digital long-term preservation is similar to the preservation of written knowledge. Many ancient texts, e.g., the bible, have been preserved without preserving the original physical writing. These texts are known in modern times because they were “migrated” (copied) in the medieval monasteries and distributed all across Europe. As long as the alphabet (greek or roman alphabet) and the language (greek or latin) are known, the text can be read and interpreted.

2.1.1 Communication with the Future

We can describe the process of digital preservation as communication with the future ([3] and [4]). To be able to pass the information, i.e., the archived digital objects into the future, we have three layers the information will pass as depicted in Figure 2.1.

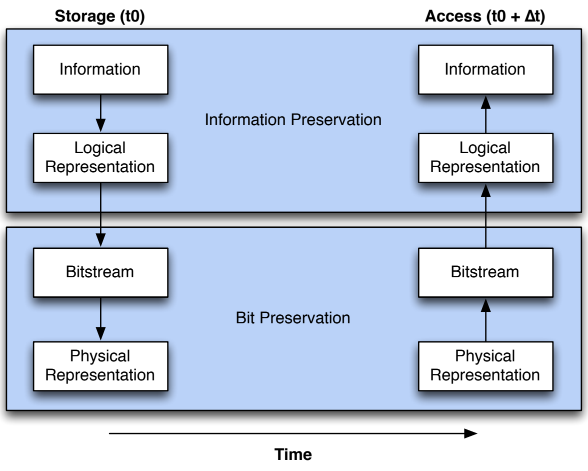

Figure 2.1: Preservation as Communication with the Future

At storage time (t_0), the information is stored as a logical representation inside a bitstream onto a medium. At access time ($t_0 + \Delta t$), to be able to access the information this conversion needs to be reversed, by accessing the medium, reading the bitstream, converting the bitstream into a logical representation, and finally converting the logical representation back to the initially stored information. Each layer is exposed to risks, that can endanger the preservation of the information.

At the lowest level, the *physical representation* of an archived digital object needs to be preserved over time. Preservation at this level is challenged by the basic decay of the physical media used for storage. For example the life expectancy for optical disks lies between 2 and 50 years (depending on quality and storage conditions), those of magnetic tapes between 2 and 30 years and for hard-drives between 5 and 10 years.

At the *bitstream* level, the preservation task is challenged by technological obsolescence of the hardware and software used to store the digital objects and which is also needed to access them. The time-span for technological obsolescence lies between 18 months ([5]) and 5 years ([6]).

At the *logical representation* level, we need to preserve the syntactic and semantic rules needed to interpret the bitstream, without which the digital data stays by itself meaningless. In order to become understandable to humans, digital data needs to be interpreted and represented by a computer system.

2.1.2 Key Aspects

The process of archiving has a much bigger meaning in such a context than just to make a permanent copy on a storage medium. It incorporates the lasting availability and with it a further usability and interpretability of the digital resources. Additionally, throughout the whole archiving process, also the integrity, authenticity, and provenance of the archived digital objects needs to be maintained.

The areas that a digital preservation solution will address are mainly the same, but there are still differences that are given by their respective Designated Communities. A designated community is the main group of users that will utilize the services of an archive. The form in which the archived information are preserved should be appropriate for those users so that they are able to understand them independently and without expert assistance.

Long-term digital preservation thus has the following key aspects that need to be addressed by any preservation system:

Preservation of the Bitstream To be able to interpret the digital object, we first need to read the data-bits. Depending on the storage type, this would generally require the following: (1) the medium is physically not damaged (e.g., no mechanical defects, or demagnetization, etc.), (2) a reader for the storage medium exists and is fully functional (e.g., tape reader, etc.), and (3) a computer is available where we can connect the reader, including the system software needed to copy the digital data from the tape to the computer.

Digital data from many heterogenous sources and on many different data carriers need to be consolidated as early as possible into one homogenous storage system.

This system needs to be functionally autonomous with one main task which is bitstream preservation. Important properties of such a system are full or at least partial automatization of the control mechanisms (e.g., integrity checking, data carrier migration, etc.).

Preservation of the Interpretability This means that not only the data itself (bitstream) and the data carrier need to be preserved to guarantee further usability and interpretability, but also the description for its interpretation and rendering by a computer system, and general meaning. As an example, to be able to interpret a word document in 200 years from now, we would also need to provide the specification for decoding the word document. To be able to do so, we need to archive additional metadata alongside the bitstream. First the r*epresentation metadata*, which will contain information that will allow the data to be rendered and used as information. Further we need *reference metadata*, which will allow us to identify and have a description of the content contained in the digital objects that we archive. Also *context metadata,* which provide us with important information regarding the purpose for the creation of the digital content or its past usage.

Preservation Quality Should we finally be able to circumvent all the pitfalls until now and manage to save the digital data and their interpretability into the future, another question arises about the integrity and authenticity of the archived digital objects. If we can not fully determine the provenance, and also have the full trail of all changes that the digital object has gone through during the preservation process (e.g., data format migration, annotations, etc.), then we have failed in our attempt to preserve it for the future.

Further more, digital archives are new institutions. As such, it is important for their trustworthiness that their institutional structure and backing are transparent and sustained, and also conform to some criteria accepted by the preservation community such as OAIS, TRAC, etc.

2.1.3 Long-Term Preservation Strategies

To be able to provide sustainable availability of digital objects, we need to address all three mentioned key aspects with suitable measures. The available strategies can be categorized into four categories: *Technology Preservation*, *Migration and Refreshing*, *Per-*

manent Medium, and *Emulation and Virtualization*. Of the mentioned measure, not all can be seen as equally practical, and generally usable (e.g., computer museum). Here, we give just a short overview of the four strategies to provide a complete picture. The following chapters are then focused on migration and refreshing strategy.

Technology Preservation (Computer Museum) In a computer museum, the digital objects and also all the technology necessary for their access are preserved in their original logical and physical form. This means that we would access obsolete documents with the original software on old machines.

The idea of computer museums is highly problematic, for several reasons [7]. Keeping old machines running indefinitely at a reasonable cost is very unlikely. Further, the old documents and the software will not survive on the original medium and would require the development of unique new device interfaces so that we could store and run the documents and software on newer storage devices and mediums. Even if this would be possible, the computer chips on the main hardware would still fail over time as they have a limited life-time.

As we can see, a computer museum is not a generally usable solution, but in some instances the only way, e.g., preservation of computer game platforms.

Migration and Refreshing Following the technological change, the digital objects will be continuously migrated to technologically current storage mediums and data formats. This would guarantee the readability of the archived digital objects with current methods and programs. Data format migration is doable with simple data objects (e.g., images, text documents, etc.) but can be a challenging task when we would need to perform data format migration on complex objects (e.g., databases, etc.). Refreshing denotes the preservation activity of transferring data between two storage mediums, an important preservation activity which needs to be performed periodically, to prevent any data loss or unintentional alterations to the data, caused by the physical degradation of storage mediums.

Permanent Medium To provide long-term preservation, the digital objects would be stored on a storage medium, which is explicitly hard- and software independent, and extremely durable. For example, a solid nickel disk as used by the Rosetta project [8]. As the access in the case of the Rosetta-Disk is by visual means, also the digital objects would need to have a visual representation to be able to be accessed from such a medium. In the following, we look at long-term preservation on microfilm as the permanent medium.

Using microfilm as a long-term preservation medium involves basically the exposition of the bitstream encoded as textual characters, barcode like images, or the visual representation if existing (e.g., PDF, TIFF, etc.).

The Arche project [9] uses a color microfilm laser writer they have developed, to expose the microfilm. To read the data, they use a special microfilm scanner. The data written on the microfilm is composed of metadata in a textual representation, and the visual document representation. This approach is only feasible for documents having a visual representation.

The PErsistent VIsual ARchive (PEVIAR) [10, 11] project uses a similar but extended approach. The data written on the microfilm consists of the visual representation (i.e., documents preview image), textually encoded metadata in a tabular and XML representation, the pseudocode containing the decoding instructions, and the digital data encoded as a 2D barcode.

Further work was done by [12], analyzing different encoding techniques for storing data on microfilm, data recovery, and the cost issues. They have concluded that microfilm-based storage is a feasible alternative for permanent storage of the digital objects.

Emulation and Virtualization In digital preservation, we can differ between the archiving of data, and archiving of the behavior exhibited by a computer program [13]. The first we solve through migration, but for the second we need emulation.

Emulation in the context of long-term preservation means the replication of the functionality of an obsolete system [6]. This ranges from the emulation of hardware platforms (e.g., using SIMH on a Linux system to emulate VAX hardware and run OpenVMS [14]), over emulation of operating systems, to application emulation. Which level of emulation is used, depends on the requirement where in the chain – original data, original software for interpreting the data, operating system for running the software, hardware running the operating system – the entry-point needs to be, to allow the long-term preservation of the intended target.

Early work on emulators was done by MIT and the Proteus system [15], and Stanford on SimOS [16, 17], although in a different context. This work was performed with the goal to better understand and improve the design and behavior of computer systems, as these emulators where denoted as simulators. The work on SIM [18] had a different orientation, with the goal of computer preservation.

The first work on emulation directed towards their usage as a preservation strategy was done with the Emulation Virtual Machine [19] and Universal Virtual Computer [20, 13].

Based on this early work, the Koninklijke Bibliotheek (National Library of the Netherlands) and the Nationaal Archief have developed a durable component-based computer emulator Dioscuri [21], specifically for the digital preservation domain. The emulator was build with two main goals, modularity and durability. The emulator is build modularly, where each emulated computer component (e.g., CPU, disk, memory, etc.) is implemented as a module. This modularity allows to create emulators for computers with different configurations, which can then be customized to represent the original hardware. Durability is achieved by building the emulator on top of a virtual machine (Java Virtual Machine).

Recent work following emulation as a preservation strategy include work on console game preservation [22], and home computer preservation [23].

2.2 OAIS Reference Model

The Open Archival Information System (OAIS) Reference Model [1, 2] is a widely accepted and used terminology to describe the various processes involved in an archiving institution. It does so by providing frameworks and concepts that are needed for understanding the long-term preservation and access of digital data. The OAIS functional model consists of various entities interacting with each other as displayed in Figure 2.2.

The general proceeded sequence inside an archive described by the OAIS reference model is the following:

1. A producer who created the content that he wants to archive provides a *Submission Information Package (SIP)* to the *Ingest* entity.
2. The Ingest entity creates from the SIP an *Archival Information Package (AIP)* and hands it over to the *Archival Storage* entity.

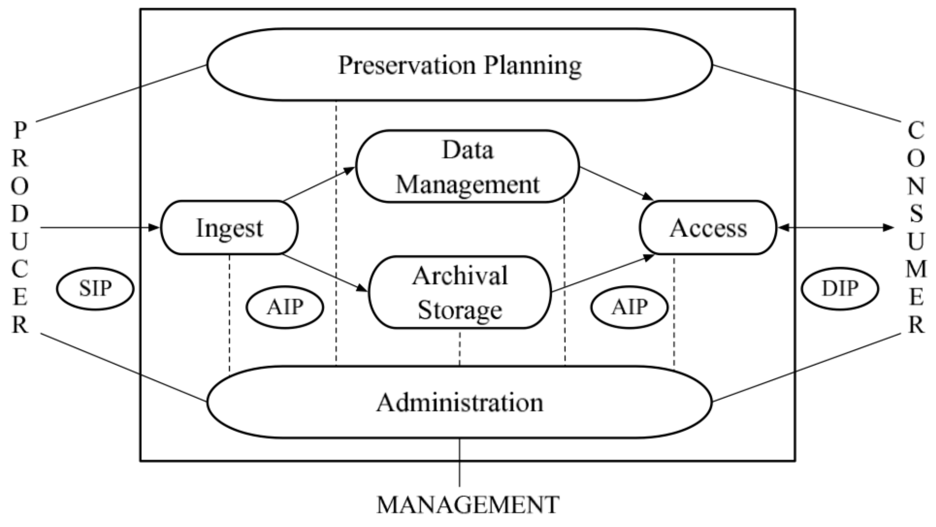

Figure 2.2: OAIS Functional Entities

3. The descriptive information related to the AIP is provided to the *Data Management* entity.
4. Next, a consumer will interact with the *Access* entity, and use the descriptive information to search and request relevant information.
5. The requested AIP will be retrieved from the Archival Storage entity and further transformed by the Access entity into the appropriate *Dissemination Information Package (DIP).*
6. All activities in the archive are carried out under the guidance of the *Administration* entity.
7. The employed preservation strategies and techniques are recommended by the *Preservation Planning* entity and are put in place by the Administration entity.

Depicted in Figure 2.3 is the *Archival Storage* entity. This entity provides services and functions for the storage, maintenance, and retrieval of AIPs. Archival Storage functions include receiving AIPs from the Ingest entity and adding them to permanent storage, managing the storage hierarchy, refreshing the media on which archive holdings are stored, performing routine and special error checking, provide disaster recovery capabilities, and providing AIPs to the Access entity to fulfill the order.

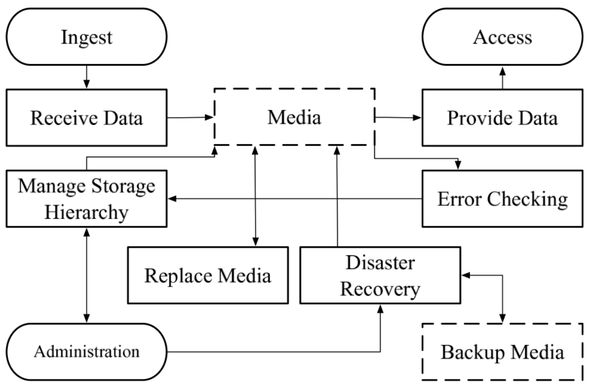

Figure 2.3: OAIS Functional Entity: Archival Storage

2.3 Digital Objects Metadata

In the following section, we provide a short overview over some of the more important metadata standards, tools for digital objects validation and characterization, and archive access API standards used in the field of digital preservation.

2.3.1 Metadata Standards

METS

The Metadata Encoding & Transmission Standard (METS) [24] is a very flexible standard defined using the XML schema language, and can be used for encoding descriptive, administrative, and structural metadata of digital library objects. For a better understanding, it is important to take a look at the structure of a METS document. A METS document can consist of up to seven major subsections [25]:

1. Mets Header (metsHdr)

2. Descriptive Metadata Section (dmdSec)

3. Administrative Metadata Section (amdSec)

4. File Section (fileSec)

5. Structure Map (structMap)

6. Structural Links (structLink)

7. Behavior Section (behaviorSec)

One major subsection is mandatory, the Structure Map. The Structure Map is the only requirement of any METS document and is often referred to "the heart of a METS document" [26]. METS provides a relatively easy method for encoding structural, administrative, and descriptive metadata for a digital library object. A typical division of a digital library object could be pages or sections that contain a book. METS has two main uses: a standardized mechanism, mainly used for exchanging digital library objects between repositories and an encoding mechanism for digital library objects. METS documents can be encoded with PREMIS metadata. This is mainly the case for exchange purposes [27]. PREMIS is a "PREservation Metadata: Implementation Strategies" which is the name of an international working group. The PREMIS Data Dictionary defines main keys that repositories should know for preservation purposes [28].

NISO MIX

NISO MIX is an XML schema which allows to use a format for storage and/or interchange of data in the Data Dictionary - Technical Metadata for Digital Still Images. The idea of this dictionary is to facilitate interoperability between services, systems and software. Furthermore, it has been designed to support the long-term management of digital image collections and the continuing access to these collections. This dictionary enables users to exchange, develop and interpret digital image files as well [29].

The current version of MIX is MIX Version 2.0.

TextMD

TextMD specifies technical metadata and it is just like NISO MIX a XML schema, too. Usually it has been used with METS but it can be a standalone document as well [30]. A schema of textMD allows the following functions: character, encoding and markup information, fonts and languages, page ordering and sequencing, processing and textual notes and technical requirements for printing and viewing.

LMER

Almost together with the development of PREMIS the German National Library began to work on an initiative for metadata which can be preserved over a long period of time. This initiative with the name "Langzeitarchivierungsmetadaten für elektronische Ressourcen" lead to LMER (Long-term preservation Metadata for Electronic Resources) [31]. The idea of LMER is to allow collecting metadata which can be utilized for the long-term preservation of electronic documents.

Dublin Core

Dublin Core (DC) exists of 15 different descriptors which were created by an interdisciplinary team consisting of librarians, content experts and so on. The goals of the Dublin Core team are: simplicity, interoperability, applicability, extensibility, adaption of standards and common semantics. The Dublin Core is great for making basic statements about resources and it is a small language [32]. The Dublin Core exists of two categories: elements and qualifiers [33].

The qualifiers are split in two groups as well: encoding schemes and element refinements. Encoding schemes are parsing rules that assist in the interpretation of an element value whereas element refinements make the meaning of an element more specific without extending its definition too much.

The Dublin Core Metadata Initiative (DCMI) maintains the Dublin Core Metadata Element Set. The 15-element set is divided into three groups:

- Content: Title, Subject, Description, Type, Source, Relation, Coverage
- Intellectual Property: Creator, Publisher, Contributor, Rights
- Instantiation: Date, Format, Identifier, Language

TEI

The Text Encoding Initiative (TEI) was created for the purpose to develop guidelines that would permit projects to share data between each other and to advance the development of tools [34]. Throughout the years, the TEI has accomplished two things: a new data description language which mainly has an influence of further and new WWW standards and it has created a research community which work with a new perspective on the understanding of the role of a text structure.

2.3.2 Object Format Identification, Validation, and Characterization

JHOVE

JSTOR/Harvard Object Validation Environment (JHOVE) [35] is a framework developed through a collaboration project between JSTOR and the Harvard University Library. The project had the goal to develop an extensible framework which would provide object validation.

JHOVE provides modules for various format types. The framework can be used to automate the process of format identification, validation, and characterization.

At run-time, JHOVE allows to be configured by specifying the desired data format modules and output handlers. The output of the tool is an XML report, where the information is formatted according to the specification of each module (e.g., JPEG2000 and TIF use the NISO MIX format).

Although JHOVE does not directly provide PREMIS output, it could be used for automated generation of preservation metadata, by combining the generated XML report with XSL transformation, and convert the output to PREMIS specific elements.

DROID

Digital Record Object Identification (DROID) [36] is a tool developed by the National Archives of the UK together with PRONOM for providing an automatic file format identification. PRONOM is an online registry, which contains technical information about file formats with descriptions for the file structure, and the need software and hardware environments for access.

DROID uses the signatures stored in PRONOM as a means for file identification. New updates to the PRONOM database are automatically detected by DROID and are downloaded as to keep the tool always up-to-date.

As with JHOVE, the report generated by DROID could be combined with XSL transformation, and the output converted to PREMIS specific elements, as a means to provide an automated generation of preservation metadata.

Use Cases

The potential use cases for JHOVE and DROID can be summarized as following:

- Identification (JHOVE, DROID)
    1. I have an object; what format is it?
- Validation (JHOVE)
    1. I have an object that claims to be of format F; is it true?
    2. I have an object of format F; does it meet profile P of F?
    3. I have an object of format F and external metadata about F in schema S
- Characterization (JHOVE, DROID)
    1. I have an object of format F; what are its salient properties?

2.3.3 Open Access API Standards

In this section we describe two API standards from the Open Archives Initiative.

OAI-PMH

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a framework which bases on metadata harvesting [37]. The OAI-PMH allows to harvest recurrently of metadata based on XML from one place to another [38], and it is based on existing standards, such as the IETF Hypertext Transfer Protocol (HTTP) and the W3C Extensible Markup Language (XML). The XML is used to encode the exchanged metadata. The framework offers an application-independent interoperability, and it has two parts [39] :

- Service Providers: The metadata harvested via the OAI-PMH, they use this metadata for building value-added services
- Data Providers: They handle systems which assist the OAI-PMH for exposing metadata

OAI-ORE

Open Archives Initiative Object Reuse and Exchange (OAI-ORE) is a standard for the exchange of compound digital objects and the description of Web resources [37]. The compound digital objects are just like aggregations, and they can merge distributed resources together with different media types such as data, video, images and text. The main goal is to give the enriched content of these aggregations along to applications which support preservation, reuse, authoring, exchange, deposit and visualization. OAI-ORE is mostly influenced by the RDF model which uses the idea of triples for describing things. The triple consists of a subject, an object and a predicate [40]. Each part of this triple is a URI, except the object, which can be a plain text value as well. OAI-ORE sets its focus on these objects and the relationships between these objects.

2.3.4 Summary

We have given a short overview of a selection of preservation metadata formats, that allow us to store different types of metadata required for the preservation of digital objects. We have further presented two tool frameworks, which provide format identification, validation, and characterization. They can also be used to automatically extract preservation metadata. At the end of this section, we have presented standardized interfaces, which can be used to open the archive to external users, and allow them to access the metadata stored in the archive over open and standardized interfaces.

2.4 Distributed Systems

In the following, we provide a short discussion regarding a few selected distributed system architectures. The discussion includes not only a short overview over the general attributes of each architecture, but also their suitability in the context of digital preservation.

2.4.1 Peer-to-Peer Systems

A Peer-to-Peer (P2P) System can be described as a distributed system which is comprised of equal peers, i.e., systems (e.g., computer nodes) which are equal in the sense of the role they perform in this network. As opposed to the client/server architecture, in P2P systems every peer performs the client (consumes services) and the server (of-

Field of use	**Network or Protocol**
File Sharing	BitTorrent, Direct Connect, eDonkey, FastTrack, GNUnet, Gnutella, Gnutella2, Kad Network, Napster, OpenNap
Software Distribution	BitTorrent
Media Distribution	BitTorrent
Internet Information Retrieval	Domain Name System
Distributed Data Store	Freenet
Peer Applications	JXTA, Windows Peer-to-peer
Video Streaming / Multicasting	P2PTV, PDTV, Peercasting
Chat/Collaboration/Social Network	GNUnet, Krawler, Pichat, JXTA
Distributed Discussion	Usenet

Table 2.1: Overview of P2P Networks and Protocols

fers services) role, depending on the task at hand. There is no central coordination for the behavior of the peers, rather the P2P architecture should be able to handle failures, and the high dynamics of large scale systems by self-organizing itself.

Peer-to-peer system have become very popular and widely known from file-sharing applications like Napster, Gnutella, BitTorrent, etc. They all have shown the power of the P2P architecture but have also brought attention to many discussed problems of technical, legal and economic natures. Books discussing these aspects of P2P systems are for example [41] and [42].

Further areas besides file sharing where P2P architecture is used are for instance collaboration and groupware applications, instant messaging, etc. Table 2.1 shows the different usage fields and the prominent networks and protocols in usage.

2.4.2 Grid Computing

Grid Computing is a form of distributed computing, where a virtual supercomputer is created from a cluster of loosely coupled computers. It was developed for solving computationally intensive problems. The difference between Grids and classic computer clusters lies in their looser coupling and geographical dispersion of the resources involved in a Grid. Also, Grids are mostly created with special applications in mind and use standardized libraries and middleware.

A first definition of Grids was provided by Ian Foster and Carl Kesselman [43]:

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

As this first definition for Grids was given before there where any actual Grid systems, a revised definition was given in [44]:

> The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization (VO).

The main difference to the initial definition lies in the addition that now the usage of shared resources is defined through Virtual Organizations (VO). The incorporation of VOs plays until today an important role in any implementation of a Grid. Also, now the Grid does not only include high-end computational resources, but any resource that is needed for the application at hand.

2.4.3 Service Oriented Architecture

Service Oriented Architecture (SOA) is a paradigm for developing distributed systems in the form of interoperating services. Each service represents a well-defined business functionality enclosed in a software component, which can be reused. Existing IT components such as databases, servers, or websites can be packaged as services, and then through orchestration composed to higher level services. The goal is to provide a long-term reduction in software engineering costs, and the rise in flexibility of the business processes by reusing already existing services in different applications.

SOA was first described by Gartner in 1996 [45, 46], but the generally accepted definition was defined by OASIS in 2006 [47]:

> "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."

Attributes

A central element of SOA are the services. There are a certain number of attributes which are ideally found in every service used in SOA. In practice, not all attributes are always present [48].

A service is an IT-representation of a business functionality, encapsulated, and can be used on its own. Further, a service is available over the network and has well defined and published interfaces. To use the service, only the knowledge about the interface is needed, and there is no need to know any details regarding its implementation. Also, a service is platform independent, which means that the service provider and consumer can use different programming languages on different platforms. Every service which is deployed is registered in a catalog. The application using a service does not need to have access to the service at creation time, i.e., services are dynamically localized and linked at run-time.

Remarks

Every SOA solution is adjusted to the requirements at hand and presents a very individual design. The communication between the services can be realized with any protocol (e.g., IIOP, CORBA, REST, JSON, SOAP, etc.), as it only functions as a transport mechanism for the actual message used in the application.

2.4.4 Cloud Computing

Cloud Computing has emerged as a synthesis out of several other computing research areas such as high performance computing, virtualization, utility computing, and Grid computing. Cloud computing describes the approach to abstract IT-infrastructure resources, e.g., CPU, storage, network, or even software, and to dynamically adapt their allocation to match the current demand, over the network.

From the users perspective, the available IT-infrastructure seems distant and opaque, like a cloud. Access to the services provided in the cloud are exclusively done over predefined APIs and protocols. The range of services provided by Cloud computing includes the whole IT spectrum, from infrastructure, over platforms, to software.

Service Models

In 2009, the National Institute for Standards and Technology (NIST), has published the first draft of definitions regarding the different service and deployment models. In 2011, the final version was published [49]. According to these definitions, we can differentiate between three different service models:

Infrastructure as a Service (IaaS) The cloud provides access to virtualized computer resources like CPU, network, and storage. The users are free in their choice of deployed software, and are responsible for their installation, management, and running.

Platform as a Service (PaaS) The cloud provides access to programming and runtime environments, with dynamic, and flexible computing and storage resources. The user develops and deploys their software solution, inside an environment, which is run and maintained by the service provider.

Software as a Service (SaaS) The cloud provides access to a collection of software applications, running on the infrastructure maintained by the service provider. SaaS is also called Software on Demand.

Deployment Models

Beside the different service models, the NIST also defined different deployment models.

Public Cloud The public cloud delivers access to the abstracted IT-infrastructure to the general public over the Internet.

Private Cloud The private cloud delivers access to the abstracted IT-infrastructure only inside their own organization.

Hybrid Cloud Is the combination of private and public access depending on the needs of the users.

Community Cloud Is a semi public cloud, where the access to the services is provided to a community, i.e., to a group of individuals, institutions, corporations, with similar interests, which share the cloud with each other.

Characteristics

The essential characteristics of Cloud computing according to the NIST are:

- The ability for the user for self-service provisioning, and the ability of the system to provide resources available as-needed.
- Scalability of the system, decoupling usage peaks and infrastructure limits.
- Reliability and fault-tolerance guarantee permanently, the defined quality standards of the services provided for their users.
- Optimization and consolidation allow adaption to current environmental standards, which can be successively introduced by the cloud provider.
- The monitoring and maintaining of the Quality of Service (QoS) can be continuously provided without any impact on, or involvement of the users.

2.4.5 Summary

In this section, we have presented a few distributed system architectures. These architectures should not be seen as being orthogonal to each other. They all share some basic common assumptions, e.g., Cloud computing builds on ideas from Grid computing, Service Grids combine SOA and Grid computing, Peer-to-Peer Services combine SOA and P2P systems, etc.

As we have seen, there are a number of different distributed system approaches. From their technical capabilities, all these approaches can be used to implement a distributed preservation environment (as we will see in Chapter 6).

Contrary to shorter lived systems, in the context of long-term preservation, we need to additionally ensure if the approach used is feasible when long-term use is a

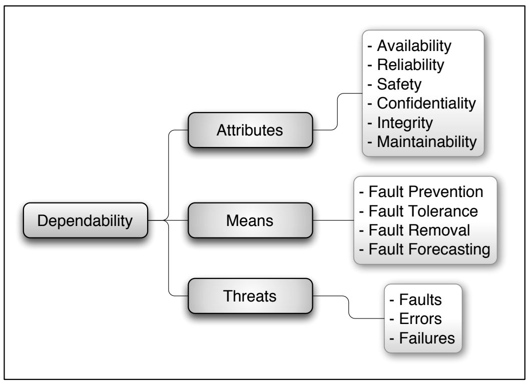

Figure 2.4: The Dependability Tree

requirement. For example, can we assume that a Cloud system provider is going to be in business forever, and does the approach used provide enough flexibility to be integrated with or exchanged for future technologies/approaches?

2.5 Dependability in Distributed Systems

An important aspect of distributed systems is dependability. Dependability is also an important aspect when speaking about long-term preservation environments, and one of the main motivators to use the distributed systems approach in the context of long-term preservation.

In a distributed system, achieving dependability is an important goal, that needs to be addressed on multiple levels. Under dependability, we understand the trustworthiness of our system in performing some specified task or providing a service [50]. There are three main parts to the concept of dependability: the *threats* endangering the dependability of a system, the *attributes* a dependable system possess and the *means* by which a system can attain dependability as depicted in the dependability tree in Figure 2.4.

As shown in the dependability tree, dependability possesses different *attributes*. *Reliability* being the ability of the system to perform the set task or service correctly when asked. *Availability* means that the system is available when asked. *Safety* is the ability to avoid excessive costs, and *Security* the ability of preventing unauthorized access to the system.

The *means* by which dependability is attained can be categorized into four major groups [51]: *fault-avoidance/prevention*, *fault-removal*, *fault-tolerance*, and *fault-forecasting*. *Fault-prevention* incorporates design methodologies consisting of techniques for design, testing, quality control, evaluation of reliability and safety, which attempt to make software provable fault-free. *Fault-removal* incorporates methods that aim to remove faults after the development stage has been completed, which is done by exhaustive and rigorous testing of the final product. *Fault-tolerance* incorporates techniques and methods under the assumption that a system has unavoidable and undetectable faults, by employing error detection and system recovery with the goal of the system to operate correctly even in the presence of faults. *Fault-forecasting* consists of evaluation of the system behavior over time.

And finally, the third part to dependability being the *threats*, which consist of faults, errors, and failures.

Fault-avoidance/prevention and fault removal are properties which are relevant during the development, testing, and evaluation stage, and will be addressed in Chapter 4 and 5. In the following we will concentrate our discussion to fault-tolerance as the main property that we can address, that will lead to higher dependability of the system during operation.

Before we go deeper into the discussion of fault-tolerance, we will need to define the terminology surrounding it. So in the following we introduce some general terminology and concepts that are behind fault-tolerance [52].

2.5.1 Faults, Errors, and Failures

The definition of fault-tolerance implies that there is a specification of what we define as correct behavior. A *failure* occurs when the service provided by a system no longer complies with its specification. The part of a system state or cause which is liable to lead to failure is called an *error*. The error itself is the result of a defect in the system. This defect as the cause of the failure is called a *fault*. Figure 2.5 depicts these basic concepts in an UML class diagram [53]. So in summary, the fault being the cause, resulting in an error, and causing failure.

When discussing faults, it is helpful to look at different classifications based on *duration*, *cause*, and *behavior* as shown in Figure 2.6.

When looking at the *duration* of faults, we can distinguish *transient* and *permanent*

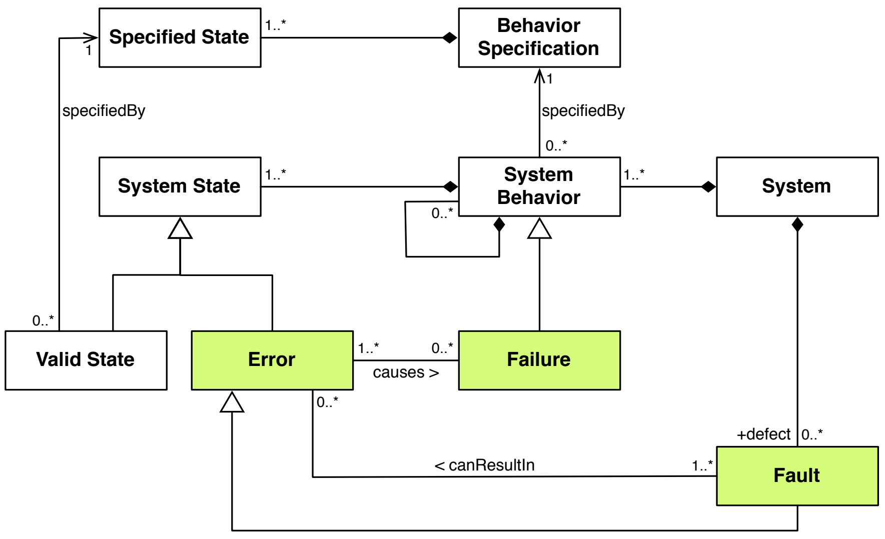

Figure 2.5: Failures, Errors, and Faults

faults, where the transient fault has a subtype of *intermittent* faults. The main difference is that a transient fault will disappear without intervention unlike a permanent fault which needs to be actively removed. From an engineering point of view, it is more complex to deal with transient faults especially with intermittent faults as they are often unpredictable.

Using *cause* as the classifier, we will have *design* and *operational* fault classes. The design being the faults introduced during the designing of the system, e.g., coding errors, etc., and operational faults being the faults that occur during the running of the system.

As a third classifier, we can take the *behavior* of components once they have failed. Here, we have *crash faults* where the component simply stopped working altogether, *omission faults* where the component fails to perform its service, *timing faults* where the component does not complete its service on time or completes to early, and *byzantine faults* which are faults of an arbitrary nature not covered by any preceding class.

2.5.2 Fault-Tolerance

In a distributed system, such as DISTARNET, the reliable functioning of the system depends on remote nodes communicating over the internet. The structure and inherent complexity of the physical world does not actually allow for absolute 100%

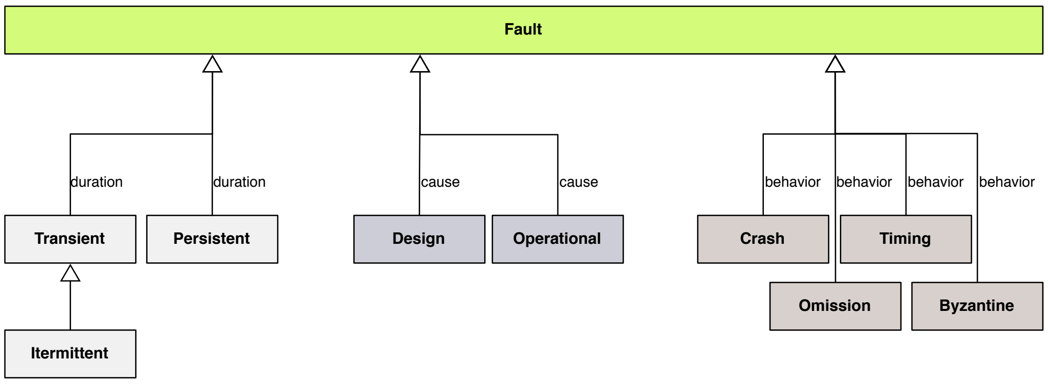

Figure 2.6: Different Classifications of Faults

foolproof software [54]. As hard as we may try, the possibility that something can go wrong can never be fully removed. We can only try to reduce the probability of failure to an acceptable level.

Fault-tolerance techniques are a means of reducing the risk of faults. *Fault-tolerance* is defined as the ability of a system to perform its function correctly even in the presence of internal faults [55, 51]. As such, the intention of providing fault-tolerance is in increasing the dependability of a system.

General Procedure

The general procedure for dealing with faults can be divided into three interrelated phases, which are *fault detection*, *fault diagnosis and localization*, *and fault elimination and recovery*.

With *fault detection,* the goal is to identify that our system is in an invalid state. To minimize the effects of this invalid state we need to identify the component which is done in the *fault diagnosis and localization* phase. Through *fault elimination and recovery* we deal with the fault itself, and remove its effects by restoring the system to a valid state.

2.5.3 Fault-Tolerance in the Context of Distributed Systems

When speaking about distributed systems, then there are additional problems that need to be taken into account such as *remote site failures*, *communication media failures*, *transmission delays*, and *distributed agreement problems*.

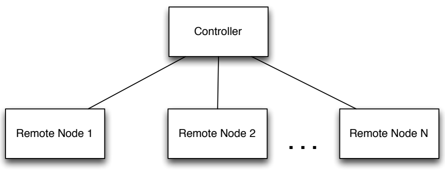

Figure 2.7: Star Pattern

Fault-Tolerant Pattern

One possible pattern for constructing a fault-tolerant distributed system is the *star pattern* (based on the master-slave pattern *[56])* shown in Figure 2.7. The distributed system consist of a *controller* and a set of *remote nodes*. The controller sends out work to the remote nodes and keeps track of the remote nodes. In the event of failure of one of the remote nodes, the controller can than initiate recovery. In this pattern, the permanent failure of the controller would lead to a global failure of the whole system. The controller is a single point of failure that needs to be made fault-tolerant. We will see later in a greater detail, how this will be dealt with in DISTARNET, by allowing remote nodes to take over the role of the controller, and by employing leader election algorithms to select the best remote node as the new controller.

2.5.4 Application Level Fault-Tolerance

As we said earlier, the techniques discussed so far like replication and redundancy are insufficient to provide fault-tolerance on the application level, and so we will now explore some additional methods that can provide fault-tolerance on the application level.

The additional methods for achieving application level fault-tolerance can be divided into two classes of strategies, namely *error processing* and *fault treatment*.

The *error processing* class can be further subdivided into two subclasses. The first, aims to remove any errors introduced into the application state, thus will implement techniques for the substitution of the erroneous state with an error-free state, and is called *error recovery*. The second, will employ methods that provide redundancy, and by doing so compensate for the error. They are called *error compensation*.

The second top-level strategy, *fault treatment*, aims to prevent the activation of faults, and so action is taken before the error arises. The two steps in this strategy

	Error Compensation	Error Recovery	Fault Treatment
Design Diversity	N-Version Programming [57], Recovery Block [58], N-Self Checking Programming [59]		
Data Diversity	Data Diversity [60]		
Environment Diversity			Restarting [61], Process Pair [62], Software Rejuvenation [63]
Checkpointing and Recovery		Checkpointing and Recovery [64, 65]	

Table 2.2: Classes of Strategies and their Fault Tolerance Methods

are *fault diagnosis* and *fault passivation*.

The nature of faults which typically occur in software has to be thoroughly understood in order to apply these strategies effectively. Table 2.2 shows the fault-tolerance methods used by these classes. All these classes are discussed briefly in the remainder of this section.

Design Diversity

To counteract and tolerate design faults that have arisen out of wrong specifications and/or incorrect coding, design diversity techniques have been developed. Here, multiple variants of a software component are developed by different teams to a common specification. Fault-tolerance is then achieved by using these variants in a redundant manner.

N-Version Programming. In N-version programming [57] $N(N \geqq 2)$ independently created functionally equivalent programs called *versions*, are executed in parallel. Then the results of all the versions are compared by a majority voting logic, and the winner is reported as the presumed correct result. This technique is used in real-life systems like railroad traffic control, airplane flight control systems, etc. The implementation cost of multiple versions and the voting logic are high.

Recovery Block. Recovery blocks [58] are analogous to cold standby in hardware fault-tolerance. In this approach multiple, functionally equivalent versions of a software component are deployed in a redundant fashion. By employing an *acceptance* *test* on every result from the primary version, the validity of the same is tested. If a result does not pass the acceptance test, then the result from the next version is tested, repeating until a result is accepted or all versions have been tried. The main difference to N-version programming is that only one version is run at a time and that there is no majority voting but an acceptance test.

N-Self Checking Programming. In N-self checking programming, [59] multiple versions of a software component including acceptance tests are developed to the same specification and are executed in parallel. Through a selection logic, the best result is being selected and reported. It is a combination of N-version programming and recovery blocks.

Data Diversity

Unlike the design diversity approaches who use multiple versions of software, the data diversity [60] approach relies only one version of the software written to a specification. The idea behind data diversity lies in the observation that software components fail sometimes for certain values of the input space and that these failures cold be averted if the input data were changed slightly so that they are still acceptable to the software component. In N-copy programming, N copies of the same program using data diversity are run in parallel with a different input set produced by a diverse-data system. The input set produced by a diverse-data system is a related set of points in the original data space. The output of such a system is selected through an enhanced voting scheme. This technique is not usable in all systems, but in systems, for example, where the sensor values are noisy and inaccurate to begin with, this technique can be used to prevent failures.

Environment Diversity

Fault-tolerance through environment diversity is based on the observation that most software failures are transient in nature and proposes that the same version of a software component is executed in a different environment [66]. Restarting was proposed [61] as the best approach to masking software faults, where environment diversity is seen as a generalization of restart.

The behavior of a software component is determined by three states, being the *volatile state*, the *persistent state*, and the *operating system (OS) environment state*. Transient faults typically occur in computer systems due to design faults in software which result in unacceptable and erroneous states in the OS environment. Therefore, environment diversity attempts to provide a new or modified operation environment for the running software.

Process Pair. The process pair approach is based on the execution of the same version of a software component on two different processors in parallel. In the event of a failure, the second processor can take over the execution. One prominent execution of this approach is the Tandems fault-tolerant computer system [62].

Software Rejuvenation. Software rejuvenation [63] is a specific form of environment diversity. It is a proactive fault management technique aimed at postponing/preventing crash failures and/or performance degradation. This technique involves occasionally stopping the running software, “cleaning” its internal state and/or its environment and restarting it. Software rejuvenation counteracts the software aging phenomenon through freeing up OS resources and by removing error accumulation. Common techniques for cleaning involve garbage collection, defragmentation, flushing kernel and file server tables, etc. Main challenge lies in the rejuvenation scheduling and granularity.

Checkpointing and Recovery

The checkpointing and recovery [64] is based on the idea of saving the state of the system, and, in the case of fault detection, recovering the execution of the system from the checkpoint where the state was saved.

Error recovery can be achieved by either forward or backward error recovery. Checkpointing can be done static or dynamic. Static checkpoints take single snapshots of the state at the beginning of a program or module execution. The system returns to the beginning of that module when an error is detected and restarts execution all over again. Dynamic checkpointing with different strategies: (a) Equidistant checkpointing uses a deterministic fixed time between checkpoints, (b) Modular where the placement of checkpoints is at the end of the sub-modular components of a piece of software right after the error detection checks for each sub-module are complete, and (c) Random where the process of checkpoint creation is triggered at random without consideration of the status of the software execution.

In [65] it is proposed and analyzed the combination of software rejuvenation (preventive fault treatment) with checkpointing and recovery to reduce the chances of activating a fault and simultaneously minimizing the loss of computation when there is a failure.

2.6 Autonomic Distributed Systems

Automation in a preservation environment is an essential requirement. Autonomic behavior would represent the next step, where the need for outside intervention is tried to be minimized.

The term "Autonomic Computing" was coined 2001 by IBM, when they released a manifesto, describing the complexity of computer systems as the main obstacle to further progress [67]. The motivation lies in the proliferation of computer devices and large software, which led to unprecedented levels of complexity, requiring highly skilled IT professionals that would perform installation, configuration, tuning, and maintenance. The mission statements define the goal to build computers that possess the ability to regulate themselves, much in the same way our autonomic nervous system regulates and protects our bodies.

In the way of IT progress, the human component is the "weakest link". The goal is to go from humans inside the (control) loop, to human *operators* "out of the loop". Even the engineers will need to be "put out of the loop", as searching for solutions will become too complex.

2.6.1 Automatic vs. Autonomic

Automatic computing is the execution of pre-programmed tasks [67]. The system works fine until something goes wrong, and at latest now human intervention will be needed. Autonomic on the other side includes self-regulation. The system response is also automatic, but modulated. The system can compensate or work around problems, so that no human intervention is needed.

If we take as an example the installation of a software upgrade. We assume that means exist to automatically deploy the needed upgrades. Also, that automatic regression tests are available, and automatic problem detection is performed. The autonomic approach would now consist of deploying, running and testing the software upgrades. Additionally, in case of a problem, the system would autonomically revert the upgrades, identify the problematic components, isolate them, and restart with the reduced set of updates.

So autonomic computing is more than a simple extension of automatic computing, and the engineering effort accordingly larger.

2.6.2 Key Properties for Autonomic Systems

Following IBM's vision, there are four main properties an autonomic system should provide [67]: self-configuration, self-optimization, self-healing, and self-protection. There are also additional properties like self-aware, self-learning, operate in heterogeneous computing environment, anticipate and adapt to user needs, but we will limit the discussion to the four main properties.

All these properties are also called self-* (self star) properties, as they begin with the term "self" in their name. This denotes their extended set of inner capabilities.

Self-Configuration

IBM's main concern is computing [67], and hence the following wish list, in which we have automatic software deployment, installation, configuration, re-configuration, and documentation. At the same time, the whole process must adhere to administrative directives, and produce reports on compliance.

The observations of IBM on the current status where that a data-center has multiple vendors, platforms, and software systems, and that the installation, configuration, and integration of new elements including hardware, software, and policies were very time consuming, and error prone. The goal was to have an automated configuration of the components according to some high-level policies, and then the rest of the system should adjust automatically.

Self-configuration can also be viewed in other contexts. Its application to networking would mean that we would have automatic configuration of addresses, and routing. In the domain of parallel computing, self-configuration, for example, would provide an automatic assignment of free processors, or computing resources in general. This could lead in very complex systems even to self-organization, e.g., finding new spacial configurations.

For our purposes, we can define that a system is self-configuring if the system is able to (re-)configure itself according to high-level policies. This property by itself does not mean that a system is autonomic, but is an important and required property for an autonomic system.

Self-Optimization

IBM's observation [67] was that there are a huge number of tuning parameters for hardware and software, for example, like the cache size, different timeout values, CPU, and bandwidth allocation, and going into the internals of software systems, ranging from the dimensioning of internal data structures like hash tables, to several algorithms that can be used for the same task, but with different profiles.

Optimization can thus be performed at many places. At design time, where the software architecture could be self-optimizing. During the implementation, for the choice of algorithms, language, and compiler optimizations. And lastly, at runtime, where the system would optimize parameters.

Although we have a large range of possibilities for optimization, the current understanding of the term self-optimization is that only parameter adaption at run-time is performed.

We can summarize, that a system is self-optimizing if the system and its components continually seek to improve their performance and efficiency. For example, in cloud computing adding more resources on demand, when some apps become too slow, in distributed computing (GRID, SOA) the automatic outsourcing of tasks, or picking the optimal connectivity subject to cost or speed. Self-optimization also includes the resolution of conflicting goals.

Self-Healing

The self-healing property involves problem solving at an analytical level [67]. The wish list for automatic problem solving thus includes that the system is not only reactive, i.e., has the ability to recover from events that have caused failure or malfunction, but also to be proactive, i.e., possess the ability to anticipate or predict failures that can happen in the future. Problem solving in general includes finding the cause, finding the cure, testing the cure, and finally deploying the cure.

The definition of self-healing by IBM is the ability of a system to automatically detect, diagnose, and repair localized software and hardware problems. It is mostly based on configuration errors, log analysis, but can also require infrastructure support, for example, when moving a set of apps and their data to a new server. As a next level beyond the definition of IBM, we can look at self-healing inside the apps, e.g., what algorithm to use, what data structure, etc. Such an extended definition can be that a system providing self-healing, is a system that asserts goal integrity over long times.

Self-Protection

According to IBM [67], a system is self-protecting when the system possesses the ability to defend itself against malicious attacks, or cascading failures, and by doing so, can prevent systemwide failures. This property is an ambitious and difficult task, as attacks are not known in advance. The main focus lies in the automatic reaction to events, and additionally employing tiered security, so that any breach on one level, will have not the potential to endanger the whole system.

In practice, self-protection can be seen as a series of steps that can be taken. First, confirm that the system has the ability to create backups, and that data resources can be recovered when needed. Next, the system monitors the network with intrusion detection capabilities, and automatically disconnects any suspicious computers in the network. Also, all security advisories are tracked, and the system verifies that all client machines have the latest patches installed.

2.6.3 Self-* Properties in General

The self-* properties are not orthogonal to each other [67]. For example, self-optimization can result in re-configuration, or self-healing can even be seen as a special case of self-optimization, in which we minimize the number of open problems. The properties are also not specific about their target, at which they are employed like at design, compile, deploy, or run-time. Also, the self-* properties are not specific about the system they are employed in, like a closed (static) set of components, or open, and distributed systems.

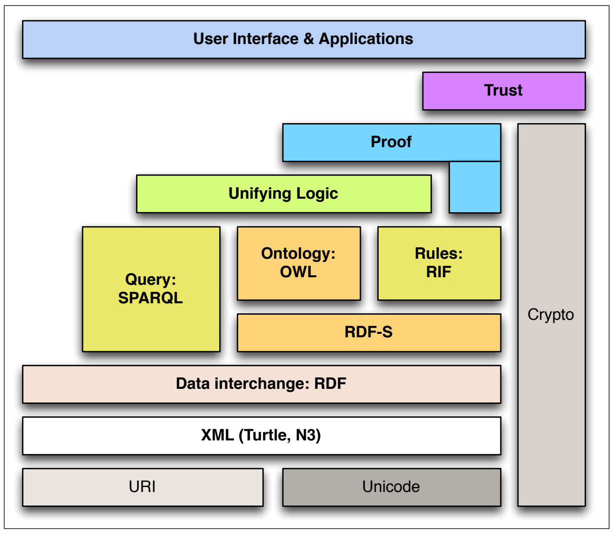

Figure 2.8: Semantic Web Stack

2.7 Semantic Web Technologies

The Semantic Web is a concept describing the further development of the World Wide Web (WWW), where unstructured and semistructured documents are prevalent, with the goal to create a "web of data", and where the computers can derive meaning from the semantically enriched data [68]. This development is lead by the W3C (World Wide Web Consortium) as the standards organization for the WWW and all the underlying standards and technologies [69].

Figure 2.8 depicts the Semantic Web Stack, the components of the Semantic Web, and how these standards and technologies stand to each other. In the following, we will describe some of these technologies in a greater detail.

2.7.1 Uniform Resource Identifier (URI)

A Uniform Resource Identifier (URI) consists of a tight sequence which identifies resources, abstract or physical. This identification is quite simple and extensible [70, 71]. The scope of an URI is global and their interpretation is consistent regardless of the context. An URI can be described as follows:

- Uniform: This has many benefits, such as uniform semantic interpretation, introduction of new types and the use of different types of resource identifiers and the identifiers can be reused.

- Resource: A resource has several uses and the term itself is used more broadly. For example resources can be abstract concepts, an image, an electronic document, human beings or even types of a relationship (e.g., "married").

- Identifier: An identifier represents the information required to distinguish what is being identified from all other things within its scope of identification.

2.7.2 Resource Description Framework (RDF)

The idea of the Resource Description Framework (RDF) is to describe logical statements about resources [72]. With RDF, it is possible to exchange, reuse, and encode metadata, and provide interoperability between applications [73]. The function of statements is to align attributes to information units and give them a value. To align these statements, it is further possible to use the RDF Vocabulary Description Language (RDF Schema) which will be explained further in the next section. It is important that the statements of the Semantic Web are described with the idea of RDF. The structure of RDF is quite simple and consists of a graph-based data model which is divided into three components which are also called RDF triple: Subject, Predicate, and Object. With the RDF triple, almost everything can be described. The resource can be a web page, an object such as a pictures, and so on.

A RDF triple connects an object and a subject and expresses this with a predicate. Figure 2.9 shows a small RDF graph, where the ovals represent resources (subject or object), the arrows represent predicates, and the box represents a literal.

Resources and literals are the main distinction inside the RDF model. An object can be a resource or a literal, whereas a subject and a predicate of a statement are always resources. Literals are simple strings, and have no further identification, whereas resources poses unique identifiers which are represented by URIs. Literals and resources can be represented as collections with three diverse types of containers:

- Alternatives: With an alternative only one value can be used with the property, from the values contained inside the container.

- Bags: Values can appear a number of times in bags. And bags are in general unordered lists, and permit duplicating values.

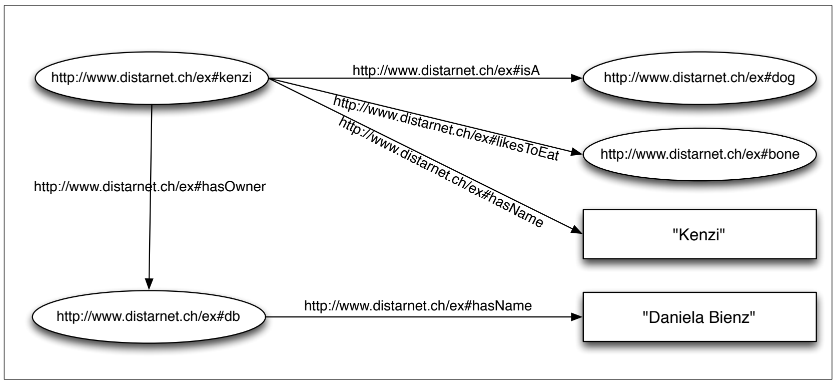
http://www.distarnet.ch/ex#kenzi
http://www.distarnet.ch/ex#isA
http://www.distarnet.ch/ex#dog
http://www.distarnet.ch/ex#likesToEat
http://www.distarnet.ch/ex#bone
http://www.distarnet.ch/ex#hasName
"Kenzi"
http://www.distarnet.ch/ex#hasOwner
http://www.distarnet.ch/ex#db
http://www.distarnet.ch/ex#hasName
"Daniela Bienz"

Figure 2.9: RDF Statement Example

- Sequences: In contrast to bags sequences are ordered lists, but duplicate values are also allowed.

As mentioned, the RDF conceptual model is a graph. For writing down RDF, the W3C specifies the RDF/XML syntax. There are a number of other serialization formats, which can be used for writing RDF such as Turtle [74], or N-Triples [75] a superset of Notation 3 [76].

2.7.3 RDF Schema (RDFS)

RDF Schema [77] allows to create simple relations between the resources and their attributes, and by doing so provides means to create taxonomies and ontologies. While the main modeling concept in RDF is the resource, in RDFS the main modeling concept is based on classes and subclasses. RDFS allows to specify ranges and applicability of property values of a class, further allowing to infer other relationships which are not explicitly stated.

2.7.4 SPARQL

SPARQL (**S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage) [78] is a query language for retrieving and manipulating RDF data stored in databases. Databases that store RDF data, i.e., triples, are also called triple-stores .

Listing 2.1 shows an example query, which would retrieve the name of all dogs from the RDF example we have seen Figure 2.9. In SPARQL queries, the Turtle RDF notation is used.

```
PREFIX ex: <http://www.distarnet.ch/ex>
SELECT ?dog ?name
WHERE {
      ?dog ex:isA ex:dog.
      ?dog ex:hasName ?name.
}
```

Listing 2.1: Example SPARQL Query

2.7.5 OWL

OWL stands for Web Ontology Language [79], a family of knowledge representation languages used for authoring ontologies, and where OWL 2 is the latest version. It extends RDF regarding expressivity, providing higher expressivity than RDFS.

2.7.6 Summary

In this section, we have provided a short overview over some of the technologies and standards of the Semantic Web. We will come back to these technologies in later chapters, where we will discuss how they can be used to provide a flexible data model for storing digital objects and their preservation metadata.

3 General Requirements and Concepts for a Distributed Archival Network

In this chapter, we provide a discussion of the general requirements for a long-term digital preservation system, and a description of the concepts and system model for our proposed solution.

We begin with a discussion of the general requirements, describing the user and system functionality needed in a digital preservation system. Afterwards, we provide a description of the main concepts and present the system model, of our proposed solution for distributed long-term digital preservation DISTARNET (2.0). DISTARNET is an acronym and represents **DIST**ributed **AR**chival **NET**work.

The scope of the described system model incorporates a core feature set of the described requirements that serve as a foundation for a preservation system, with the ability to be extended in the future to cover the full range of requirements.

DISTARNET (2.0) is a follow-up project of the former DISTARNET (1.0) [80] project. The original DISTARNET (1.0) system was developed at the Imaging and Media Lab, University of Basel and granted by the Swiss National Science Foundation[1], project number: 100012-105714. For the reminder of the document, we use DISTARNET when speaking about DISTARNET (2.0), and explicitly say when we refer to the original DISTARNET (1.0) project.

During the description of the system model, we use DISTARNET as part of the name of some of the presented concepts. The description of the system model that is presented in this chapter will be implementation technology independent. In Chap-

[1]`http://www.snf.ch`

ter 4 we then present the system architecture and implementation of a DISTARNET system, which is based on the concepts described in this chapter.

3.1 General Requirements for a Long-Term Digital Preservation System

Based on Section 2.1 and 2.2 where we introduced the basics of long-term digital preservation including the OAIS Reference Model, on TRAC (Trustworthy Repositories Audit & Certification Criteria and Checklist) [81], and on the experience from the original DISTARNET (1.0) project, we now describe a set of requirements that will ideally all be met by a distributed long-term digital preservation system. These requirements are described in very broad and generic terms, without ties to any specific technology.

3.1.1 Information Object

We use the term *Information Object* to denote a digital object that we want to archive, which consists of the digital data (bitstream) of a data object (e.g., image, audio/video, text document, etc.) and all surrounding metadata. The surrounding metadata includes representation information, reference information, and context information. It also can include additional description information for an information object (e.g., annotations), the descriptions of relationships between information objects (e.g., links), or information about membership in collection / subcollection sets.

3.1.2 User Functionality

The users of a long-term preservation system must be able to fulfill a certain number of tasks, which we will describe in the following. From the user perspective, when using a preservation system, there are some basic features that need to be provided. The user will want to be able to add content to the archive (*Ingest*), work with the content in the future (*Access*), and define some preferences regarding the long-term storage (*Preservation Planning*).

3.1.2.1 Ingest

The users of an archiving system must be able to ingest Information Objects into the system. As the Information Object being composed of the digital data (bitstream), their metadata (representation information, reference information, context information) and optional additional metadata (e.g., links, annotations, collection / subcollection membership) can be available in virtually any format. An archive may define policies, on how data are to be ingested including what representation information need to be included, and what formats can be used. Thus, a policy for ingest should be agreed between the Content Provider and the Preservation Manager. The ingest policy details the assembling of the Information Object for submission to the long-term archive.

The ingestion of data must be executed in a reliable manner, and support for a fault tolerant and recoverable ingestion provided, subject to policies between the Content Provider and the Preservation Manager who can define acceptable limits.

Further, the ability for external systems to process submissions should be provided. It should also be possible to notify these external systems about the submission of new data. Also, the system should allow to be queried regarding the status of the ingest of the digital objects.

All ingested data must conform to the ingestion policy, and as such the system must be able to verify and validate the ingested data regarding their conformance to the ingestion policy. Also, all digital rights must be preserved during the ingest.

Over time, new versions of digital objects and the representation information can be available. Thus, the system must have the ability to accept and manage new versions of the digital object and representation information updates. Also, the system must be able to accept withdrawal of digital object.

The here described ingest functionality is equivalent to the Ingest functional entity described in the OAIS Reference Model.

3.1.2.2 Access

After Ingest, the second most important feature of an archiving system is the ability to give the users of the system Access to the archived objects. As the preservation environment is not intended to be a dark archive, it should provide their users with the possibility to search for and retrieve digital objects. The users will be able to search for content either by providing free text keywords or by specifying meta-data attributes. After finding the information object the user is interested in, he should be able to have access to the digital data, metadata, and all additional metadata contained in the information object.

Access policies for the data and representation information may exist. User access is restricted to certain users and user groups on per digital object level. Sometimes an institute running an archive does not hold the license for all material that is stored within the archival storage. In these cases, the consumer is directed to the license owner's system when he wants to retrieve a particular object. An infrastructure provider might enable different institutes to store their collections at their site, and share some of those collections with each other. The system may have to provide secure interfaces for remote access (search and retrieval) to other institutes/consumers.

With time, the knowledge base of the designated community may undergo certain changes. This may lead to a situation where users or a Preservation Manager may want to enhance an existing information object with further knowledge in order to keep that specified object understandable, and up-to-date.

This described access functionality is equivalent to the Access functional entity described in the OAIS Reference Model.

3.1.2.3 Annotations, Links, and Collections

As seen in the description of the access functionality, we need to be able to enhance existing information objects that are stored inside the preservation environment. So the archiving system, and also the information objects, need to possess the ability to be enhanced with additional information. This additional information can be user created annotations attached to information objects, created arbitrary links between information objects, or information objects organized into collections and subcollections.

3.1.2.4 Preservation Planning

The individuals, institutions, or corporations can have their own policies regarding different aspects surrounding the task of long-term preservation of their materials. These policies can be driven by preservation needs (e.g., keep a minimum of three replicas on different sites, etc.), or even by some regulatory requirements (e.g., minimum geographical distance between two replicas, only allowed to store replicas inside the country, etc.). The administrators of the archive (Preservation Managers) will thus need to be able to create, manage, and assign preservation policies and plans inside the archiving system.

These preservation plans and policies also include actions, and procedures in the event of data format obsolescence relevant to the archived information objects. The archiving system should support the administrators by providing the ability for automatic detection of future data format obsolescence, and to provide data format migration capabilities.

The described preservation planning feature is equivalent to the Preservation Planing functional entity described in the OAIS Reference Model.

3.1.3 System Functionality

In Section 2.1 we described long-term preservation as a form of communication with the future. For the communication to be successful, the moment information objects are ingested by the user into the preservation environment; the preservation system needs to fulfill a series of tasks subject to user defined policies, with the goal to be able to serve these archived information objects in some distant future when the user wants to access them.

These tasks have as their goal to preserve information objects, despite potential changes of the formats in which objects are stored, and the underlying hardware environment. Therefore, a software system for digital long-term preservation has to support preservation processes that guarantee i.) *integrity*: the information captured by data is not altered in any way; ii.) *authenticity*: provenance information is properly linked to each stored object by means of appropriate metadata; iii.) *chain of custody*: location and management controls are tracked within the preservation environment; iv.) *completeness*: everything that was initially stored is also available in the future, and finally v.) *ease of access*: the necessary means are provided to find and identify the stored digital objects.

Moreover, an additional and essential requirement for a long-term preservation systems is their capability to do the necessary maintenance and failure recovery in an *autonomous* way, e.g., to identify automatically when a replication level defined through a preservation policy is no longer achieved, and to trigger corrective actions (deploy new replicas) without human intervention.

We have seen that the deployed preservation policies can include a wide range of requirements, ranging from the minimum number of replicas, to requirements regarding geographical distribution of the archived materials. The requirement for geographical distribution, paired together with the need for adequate computing and storage resources on one side, and the limited available resources of the institutions running an archive on the other side lead us to the need for a collaborative distributed archiving solutions, which provide the needed flexibility.

A distributed archiving solution for information objects is comprised of a number of nodes, building a network of cooperating nodes which provide the needed services. All nodes inside the network must be equal in regards to their functional capabilities and differ only in terms of processing power and/or storage space. There must be no central management system for the network.

The system must be able to allocate dynamically resources, i.e., storage space, for the information objects that are ingested into the network. Through dynamic replication of the information objects, the system must also provide secure and automated redundant storage of data, strong emphasis should be placed on a high storage capacity of the system as a whole. Through access rights management, access to archived information objects must be restricted accordingly.

3.1.3.1 Replication and Distribution

To provide a secure environment for long-term storage of the archived information object, a system needs to store more than one copy of each information object, i.e., create additional replicas. The created replicas should not be stored in one place; rather they need to be geographically distributed as a complete failure or destruction of a storage location should not endanger all replicas. Thus, the solution needs to provide some form of a replicated and distributed storage environment.

3.1.3.2 Fault-Tolerance and Failure Management

Local geographical disasters like fire or an earthquake has the potential to destroy parts of the preservation environment. As time goes by, hardware or software components may fail unexpectedly while storage media tend to lose bits of information.

Any abnormal conditions or problems that may harm the workings of the system must be recognized, and recovered from. This should be done by providing trigger routines that are executed every time, after a particular event that has occurred is been detected, e.g., the copyright of a digital object expires, data corrupt, etc. Any reaction to such triggered events should be executed autonomously by the system, by initiating counter measures.

The failure of one or more components in a distributed preservation system should not endanger the whole system, and should only have isolated effects. Failure (e.g., power failure, hardware failure, etc.), or disaster (e.g., natural disaster, fire) resulting in destruction, or corruption of some of the stored information objects, should not lead to a complete loss of the archived data. Automated replication mechanisms should maintain a minimum number of geographically dispersed replicas (number and location defined by preservation policies) of the stored information objects.

Any data loss event should trigger automated recovery processes that will reestablish the minimum number of geographically dispersed replicas. This should be done by either using the repaired failed storage nodes, or by using other available and suitable storage nodes found through resource discovery.

3.1.3.3 Management of Complex Information Objects

Any information objects stored inside the preservation environment are complex by nature. The complexity arises out of the requirement for preserving additional supporting information beside the bitstream of the archived data object. Even a simple digital object, like for example, an image, will become a complex information object, as we bring together and archive the digital files containing different data formats of the same image (bitstream), the descriptions of the different data formats (representation information), the descriptions of what information the image is carrying (content information), and others like reference, context, fixity, provenance, and many more.

So, the long-term preservation of digital data requires the management of complex information objects, i.e., information objects that are comprised of or are part of other information objects.

The challenge lies in the automated management of such complex objects in a distributed setting. Preserving the integrity of complex objects is a twofold problem. First, the integrity of the referential information needs to be maintained (e.g., are all references properly defined, and are all objects referred to available), and second, the integrity of the objects themselves. Referential and object integrity checking (through the use of fixity information stored with the complex object) needs to be automated. Any loss of integrity needs to trigger automated processes that will restore the integrity of the information object. If the information object cannot be repaired solely by the information it carries itself, other remote replicas need to be used. As an example, through hardware failures some information objects might become corrupted or destroyed. The information object representing a collection is partially corrupted while some of the objects that are part of the collection are completely destroyed. In this case, the discovery and subsequent recovery of the referential information (and through inference also of the endpoint information objects) need to be automated. Furthermore, integrity is an important challenge in the synchronization of information objects that are changing during the preservation process (e.g., annotations, links between information objects, collection / subcollection information). A system needs to make sure that such changes do not break (falsely) the integrity of the information objects.

3.1.3.4 Scalability

The omnipresent digitality of our culture and civilization leads to an ongoing rise in production of digital data that need to be archived. This constant growth of the volume, requires scalable distributed preservation solutions that should work efficiently even with an increasing number of users, and quickly growing volumes of data that need to be archived. The addition of storage resources should enhance the performance of the system. This requires that the processes supporting the archiving operations be automated and scalable themselves.

3.1.3.5 Openness and Extensibility

A long-term preservation system should be running per definition for a long time. During that time, it is likely that the imposed requirements on the system can change. As such, a long-term preservation system should provide clearly separated, and publicly available interfaces, to enable easy extensions to existing components, and the possibility of adding new components. The system should be able to be adapted to arising new challenges, by allowing curators of the archived information objects to specify new processes to cope with additional challenges (e.g., novel data formats).

3.1.3.6 Resource Discovery and Load Balancing

In a distributed preservation system, where each node is potentially run by a different organization, and there is no central coordinator, the discovery of newly available resources, together with the monitoring and management of existing resources is very important, and should be handled efficiently. The information gathered is important for the functioning of the processes that provide automated replication of the information objects to suitable remote storage nodes (constrained through preservation policies). The system should be able to distribute the replicas among the available resources for improving performance, where performance incorporates measures such as availability, access speed, higher security, and reliability. Dynamically incorporating new resources or correctly handling the loss of existing resources (temporarily or permanently) should be provided via automated processes. For an efficient usage of the available resources, tasks (e.g., data format migration) should be executed (immediately or deferred baring the availability of the resources needed).

3.1.3.7 Authentication, Authorization, and Auditing

A variety of different user roles and user groupings can exist in the environment of the archive, where one individual may act in multiple user roles. Access to resources should be secured to ensure only known users are able to perform allowed operations. Apart from these usual security precautions, in a distributed preservation environment (e.g., different institutions cooperate and share storage space) only the institution should be able to access and manage its owned data. No access should be possible to foreign data hosted on the local node for redundancy reasons. Cooperating institutions should be able to access other institution's meta-data and be granted access to the content of interest after having been authorized by the data owner.

The system must further support error reporting, and provide logging. All operations and their results must be recorder in a searchable and preservable manner to allow verification and validation of authenticity and chain of custody. Also, all operations should be able to be verified and validated.

3.2 Distributed Archival Network Concepts

In the previous section we have looked at the general requirements for a long-term preservation system, broadly describing the user and system functionalities such a system should possess. In the following, we describe the concepts and the system model a long-term preservation solution, which meets these user and system functional requirements on different levels.

In the following, we begin by first giving a high level overview of the DISTARNET system model. We identify the components that are part of the system model, the externally visible properties of those components, and the interactions and relationships (e.g., the behavior) between those components. After the high level overview, we take a deeper look into the system model by describing the DISTARNET processes and the DISTARNET data model.

The DISTARNET system model is designed as a representation of a fully distributed system consisting of a network of equal nodes. Figure 3.1 depicts the system model of a *DISTARNET Node*. Every node has the same buildup, i.e., is equal, making a network of DISTARNET nodes, a fully distributed network without any nodes acting as super peers.

3.2.1 Network

The DISTARNET system model represents a fully distributed system consisting of collaborating sites that deploy nodes. To provide security regarding content access we organize the sites together into *Virtual Organizations (VO)*, much like in Grid-computing. In a virtual organization, each participating institution is able to define their access policies as they see fit so that access restriction management can be used do define the allowed access to the stored content. To provide full distribution, and no single point of failure inside the virtual organization, we structure the network inside the Virtual Organization in a self organizing fashion, much like in Peer-to-Peer (P2P) networks. The so created P2P structured VO constitutes what we call a DISTARNET Sub-Network (DSN). We refer to the created network of nodes that can be constituted of multiple sub-networks, as the DISTARNET Network. As illustrated in

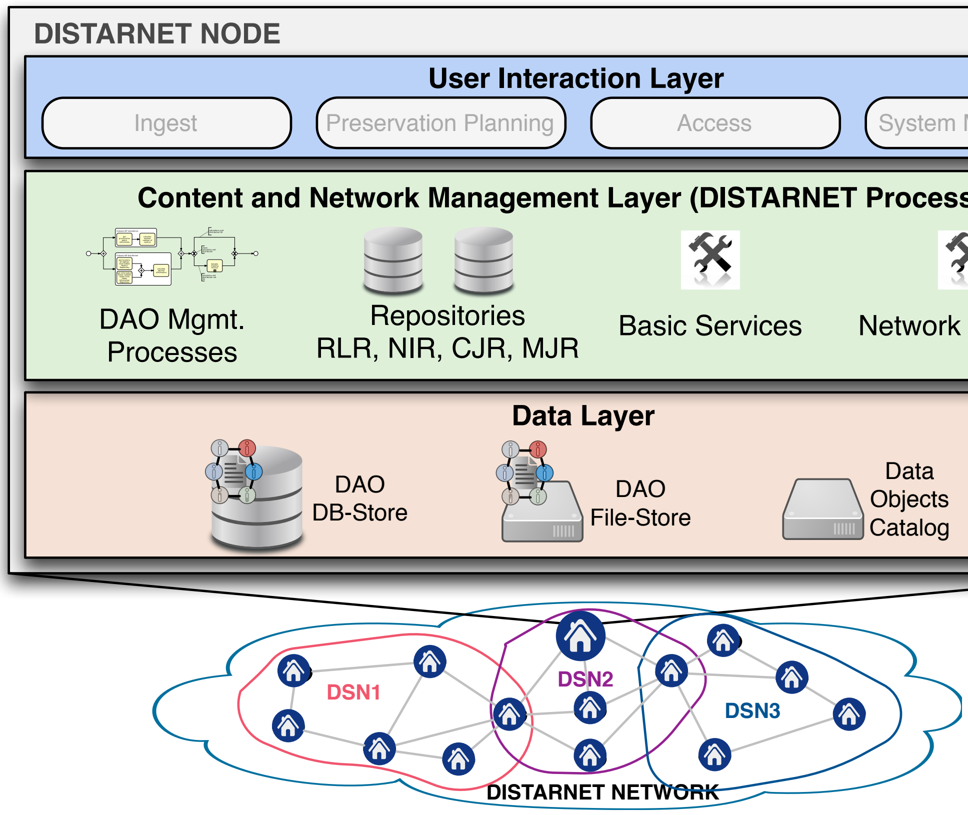

Figure 3.1: DISTARNET 2.0 System Overview

Figure 3.1, we see a larger DISTARNET Network consisting of multiple independent DSNs. A node can be part of one or more DSN if there is a need. The resources are partitioned and can only be accessed by sites within the same DSN.

Due to the P2P and VO nature of a DSN the resulting distributed system is free of any single point of failure, while providing at the same time a high degree of scalability (both in terms of nodes, users, and volumes of content to be archived), availability (through replication), and access security (through Virtual Organizations).

3.2.2 Node Layers

A DISTARNET node is based on a layered design consisting of three layers: the (1) *Data Layer*, where the different metadata representations and data catalogs are located, (2) the *Content and Network Management Layer*, where the preservation logic of the system in the form of processes, data repositories, basic-services, and network services is housed, and (3) the *User Interaction Layer* allowing user and administrative access to the system.

Data Layer

The *Data Layer* houses the *DAO DB-Store*, the *DAO File-Store*, and the *Data Objects Catalog*. This layer will be used for the storage of *DISTARNET Archival Objects (DAOs)*[2], which are a DISTARNET specific realization of the *Information Object (IO).* As mentioned, an information object encompasses the archived digital objects (e.g., image, audio/video, text document, etc.), corresponding metadata (e.g., annotations), relations (e.g., links, sub-/collections) between other information objects, and properties (e.g., access rights, availability requirements, etc.).

Content and Network Management Layer

The Content and Network Management Layer houses all the digital preservation logic, and data repositories needed for the running of the system.

The digital preservation logic consists of DISTARNET processes, and basic-services used during the execution of these processes. The design of this layer allows a reliable and fault-tolerant execution of the processes and archival management of DAOs in a distributed setting.

This layer also houses the different repositories, like the distributed *Replica Location Repository (RLR)* where the locations of all replicas are stored, distributed *Node Information Repository (NIR)* where information such as location, country, free/used space, etc. of nodes in the network are stored, the *Copy Job Repository (CJR)* used for storing information about pending remote replica creation jobs, and the *Migration Job Repository (MJR)* used for storing information regarding data migration jobs. The RLR and the NIR are built and updated with information gathered about other nodes through the *State Dissemination Process* which runs periodically on every node in the DISTARNET Sub-Network (DSN).

[2]See Section 3.6

User Interaction Layer

The User Interaction Layer is the layer over which all external interaction (user and administrative) with the system is provided. Basically this layer represents the user-interface that the users are provided for working with the system.

3.2.3 Processes Overview

The goal of the proposed system model is to posses a certain flexibility and adaptability to possible changes in the future, stemming from the long-term nature of an archiving system. Based on this reasoning, the main functionality of the system described in the system model, is provided by *processes* that consume *basic-services* during their execution, similarly to a Service Oriented Architecture (SOA) approach, with the difference that the processes taking the role of service consumers and the service provider are situated on the same node. Contrary to a monolithic approach, our proposed solution consists of a collection of processes and services. An overall application of the separation of concerns paradigm, provides us with smaller well defined processes that have a clear separation in functionality, and small reusable services. Such a design will serve us better in providing an overall flexible and adaptable solution.

The DISTARNET processes are defined in and managed by the *Process Execution Logic (PEL)*, which is housed in the *Content and Network Management Layer* of a DISTARNET node. The PEL is responsible for the execution (timer or request based), and monitoring of the execution of each process.

The goal of DISTARNET is to provide dynamic replication, automated consistency checks, and recovery of the archived digital objects, utilizing autonomic behavior and predefined processes, governed by preservation policies, and without any centralized coordinator in a fully distributed network. To achieve this, the system will exhibit certain self-* properties[3] (see Table 3.1) which we describe in the following in greater detail.

[3]For a general discussion on autonomic systems, see Section 2.6

Self-Configuration
Node Joining Process (NJP)
Periodic Neighbor-Node Checking Process (PNCP)
Automated Dynamic Replication Process (ADRP)
Self-Healing
Periodic Integrity Checking Process (PICP)
DAO Repairing Process (DRP)
Node-Lost Process (NLP)
Reliable Copying Process (RCP)
Data Format Migration Process (DFMP)
Self-Optimization
Resource Discovery
State Dissemination Process (SDP)
Parameter Optimization

Table 3.1: Main DISTARNET Processes Categorized by their Self-* Properties

Self-Configuration

In DISTARNET self-configuration manifests itself in the ability of the system to detect automatically changes in the network. Events such as new nodes joining or nodes leaving are being constantly monitored, and taken into account. Processes involved are the *Node Joining Process*, the *Periodic Neighbor-Node Checking Process*, and the *Automated Dynamic Replication Process*.

Node Joining Process (NJP): A node joins the network after the node credentials are configured, and the members of the DISTARNET Sub-Network (DSN) are added to the Node Information Repository.

Periodic Neighbor-Node Checking Process (PNCP): Every node will check periodically its neighbors, where the neighbors are members in the same DSN, by sending a message to which the receiver has to reply in a defined time. If the receiver does not reply after some defined period of time, then this node is marked as lost, and leads up to the triggering of a *Node-Lost Event*. Following this, the system begins with the self-healing behavior by starting the *Node-Lost Process (NLP)*.

Automated Dynamic Replication Process (ADRP): The ADRP is responsible for finding suitable storage nodes within the DSN, estimate the optimal number of replicas, and initiating the creation and distribution of replicas. For this, the ADRP finds – using the Node Information Repository – suitable geographically dispersed nodes for storing the replicas by taking into account possible geographical restrictions imposed by the *Preservation Policy*.

The system estimates the optimal number of replicas needed by taking into account the availability of nodes (based on statistics on the individual availability collected in the past) used to store a DAO, and via the *Preservation Policy* imposed availability threshold of the DAO itself. This estimate is used to raise the number of replicas if needed. To optimize the access performance, the system creates if necessary additional replicas by analyzing the usage patterns of the digital objects. After evaluating a DAO regarding its overall availability in the network, ADRP initiates if needed the *Reliable Copying Process* and creates new replicas.

Self-Healing

Due to the continuous monitoring of nodes, content, and processes, the DISTARNET system detects abnormal conditions or problems that may harm its proper functioning and is able to recover automatically from those situations, by means of predefined processes. The system is designed as a *fault-tolerant* system with detection and recovery mechanisms for occurrences of failures on the infrastructure, content, or node engine level. The main processes are the *Periodic Integrity Checking Process*, *DAO Repairing Process*, *Node Lost Process*, *Reliable Copying Process*, and *Data Format Migration Process*.

Periodic Integrity Checking Process (PICP): The PICP checks periodically the integrity of every DAO. In the case that a loss of integrity is detected, the process will initiate the DAO Corrupt Event, which will trigger the *DAO Repairing Process.*

DAO Repairing Process (DRP): The DRP analyzes the corrupted DAO, and initiates the RCP to get the fresh copies of the corrupted DAO from remote replicas, which are then used for the repair.

Node-Lost Process (NLP): The NLP is executed automatically in the case of a Node-Lost Event. This process analyses the lost node, and checks if the current node is the next node in line for taking over the responsibility for the management of the DAOs that where ingested on the lost node. In the case that the local node is indeed now the responsible node, it will create local replicas of the DAOs belonging to the lost node if needed, and also initiate the ADRP for these DAOs.

Reliable Copying Process (RCP): The reliable copying process is a transfer mode that uses existing replicas in the network for creating new ones in a secure and efficient manner. At process level, transactional semantics according to the model of transactional processes [82] are applied.

Data Format Migration Process (DFMP): The DFMP is used to migrate the data format of the bitstream data of a DAO, in the case where the readability of the data is endangered through data format obsolescence.

Self-Optimization

All the mentioned properties until now can only be provided if the system has the needed information on which it can act upon. As a consequence, the DISTARNET system must know its environment, especially the available resources and track their changes over time. This knowledge is provided by the *State Dissemination Process* and is used to manage autonomously and maintain resource allocation through the *Automated Dynamic Replication Process* (e.g., finding suitable nodes where data can be replicated to, automatic policy-based geographical distribution of data, etc.), and other processes needed for the operation of DISTARNET.

State Dissemination Process (SDP): The dissemination of the information needed for operating a node, which are stored in the local data repositories, is done by using periodic direct communication between nodes in the network. Periodically the SDP sends out and receive changes from other nodes in the DISTARNET Sub-Network which is then used to update the Node Information Repository and the Replica Location Repository.

Adaptive Parameters: As mentioned earlier, the ADRP and the PICP are also processes that are triggered periodically. The parameters that trigger these processes will be adapted dynamically by the system so that the time intervals between the triggering of those processes can be changed. They will be prolonged in the case that for a longer period of time there where no changes in the network, or shortened if there where recent changes.

3.2.4 DISTARNET Processes and OAIS

The DISTARNET processes correspond to the OAIS model of Archival Storage which we have introduced in Section 2.2, and are described in Table 3.2.

OAIS	DISTARNET 2.0
Ingest / Receive Data	Ingest Process
Manage Storage Hierarchy	Automated Dynamic Replication Process
Error Checking	Periodic Integrity Checking Process
Replace Media	Reliable Copying Process
Disaster Recovery	Reliable Copying Process

Table 3.2: Comparing OAIS-Archival Storage to DISTARNET 2.0 Processes

3.3 Failure Classification and Fault-Tolerance

In Section 2.5 we have introduced the notion of *Dependability* and the properties leading to it, where we have come to the conclusion that *Fault-Tolerance* is a property that needs to be an essential part of the design, when speaking about dependable systems. So in the following we discuss *Fault-Tolerance* in the context of our proposed system model for distributed long-term preservation and identify how and at what level it will be provided.

When discussing fault-tolerance in the context of our system model, we can look at and classify the faults, we are concerned about, on three distinct levels. Starting at the macro level, we have the *Distributed Infrastructure* class, where anything that endangers the distributed nature of the proposed solution is enclosed in. Next, the

Content class, where all faults that can lead to the loss of the archived data are bundled. Finally, at the lowest level, the *Node Engine* class, under which all the faults are subsumed, that can happen during the execution of the processes running on a node.

The self-healing property of the DISTARNET processes described in our system model is directed at encountering these three classes of faults. Table 3.3 provides an overview of the classes, the failures that can results from them, and the main processes that are involved in their detection, confinement, recovery, and treatment.

Fault Class	**Failure**	**Detection / Reaction**
Distributed Infrastructure	**Hardware Problems** Failure (power, hardware, etc.) Disaster (natural, fire, etc.) => *Node-Loss*	***Modules Involved:*** DP Logic ***Detection:*** PNCP ***Reaction:*** Node-Lost Event; Repository updt, ADRP
	Network Problems intermittent/periodic connection loss, etc. => *Node Dependability*	***Modules Involved:*** DP Logic ***Detection:*** PNCP ***Reaction:*** Repository updates, ADRP
Content	Localized hardware problems, malicious acts, etc. => *DAO Corruption*	***Modules Involved:*** DP Logic, DAO Storage ***Detection:*** PICP ***Reaction:*** Repository updates; DAO update; ADRP
	Format obsolescence => *DAO Representation Unreadable*	***Modules Involved:*** DP Logic, DAO Storage ***Detection:*** PICP ***Reaction:*** DFMP
Node Engine	A problem occurring during the execution of a DISTARNET process => *Process Execution Failure*	***Modules Involved:*** DP Logic ***Detection:*** process execution logic ***Reaction:*** execution of corresponding recovery process

Table 3.3: DISTARNET Fault Classes, their Effect, Detection, and Recovery Actions

As we have seen earlier, fault-tolerance is a mechanism used to provide system functionality complying with the specification in spite of the presence of faults. On the whole, fault-tolerance frameworks are focused on physical systems and not on software systems, and most applied techniques are based on replication and redundancy, and recovery. Replication is employed in the recuperation of system functionality by means of duplication of each of its functionalities in the form of replicas, where in the case of a fault another replica takes control.

In the context of the system model, employing replication and redundancy will only provide fault-tolerance to the first two classes of faults. For the third class of faults, the *Node Engine* class, other techniques are necessary since it is insufficient to add simply redundancy, as by doing so would simply duplicate the problem.

3.3.1 Infrastructure Faults

The infrastructure faults class contains all faults that can compromise the functioning of the DISTARNET Sub-Network (DSN). The DSN is the infrastructure, the backbone of our proposed solution, which in itself needs to be fault-tolerant.

Node Loss

Through general failure like power, hardware, etc., or through a disastrous event like flood, fire, etc., a node can get compromised to the point that it is not working any more, i.e., a node is lost. The loss of a node is detected through the PNCP. After a predefined amount of time that a node is not responding, its status will be set to a *Lost Node* and the *Node-Lost Event* will be triggered. The DISTARNET system will then automatically react and initiate countermeasures in the form of the *Node Lost Process (NLP)*, by reevaluating the DAOs affected by the disappeared node by the ADRP, and if needed create new replicas so that in the *Preservation Policy* defined redundancy and availability requirements are upheld again. Through predefined responsibility chains (an ordered list of the next responsible node in case of node loss), a new *responsible node* will be selected for the continued management of the now orphaned DAOs, who will then initiate the ADRP for these DAOs. This new responsible node should have information stored in the Replica Location Repository as this information is shared with all members of the DSN.

Node Dependability

Intermittent or periodic connection loss will be detected by the PNCP which logs all successful and unsuccessful communication attempts. Unsuccessful communication attempts will have the consequence that the dependability of a node is downgraded, which in itself is a measure used by the ADRP in the selection of remote storage nodes.

3.3.2 Content Faults

The content fault class contains faults associated with the content, i.e., the data, which our goal is to preserve.

DAO Corruption

Through localized hardware problems, e.g., disk crash, through improper handling of the data, or through malicious acts, the integrity of a DAO can be compromised. Periodic integrity checks are done by the *Periodic Integrity Checking Process (PICP)*, and if integrity is breached, it will automatically trigger countermeasures like finding healthy replicas in the DSN, and by using the *Reliable Copying Process (RCP)* to get remote healthy copies, and use them to repair the corrupted DAOs, to rectify the problem.

DAO Representation Unreadable

This failure is caused by the obsolescence of data formats, which can prevent the representation of a DAO to be read. To prevent this from happening, the data formats of the archived data objects are constantly monitored and warnings are issued if a given data format is becoming obsolete as defined in the *Preservation Policy*. The *Data Format Migration Process (DFMP)* can be initiated to migrate the obsolete data formats by following a predefined migration path.

3.3.3 Node Engine Faults

This third class of faults, *Node Engine Faults* that concerned about, contains faults that can emerge during the execution of the DISTARNET processes, during their interaction with the different parts of the system, or the interaction of any part of the system with another. This discussion will be continued in Section 4.1.1 in a greater depth as it is tightly coupled with the implementation, and the therein used system architecture.

3.4 DISTARNET Modules

Following the system model overview and discussion about failure classification, we will now take a deeper look into the system model and the individual building blocks.

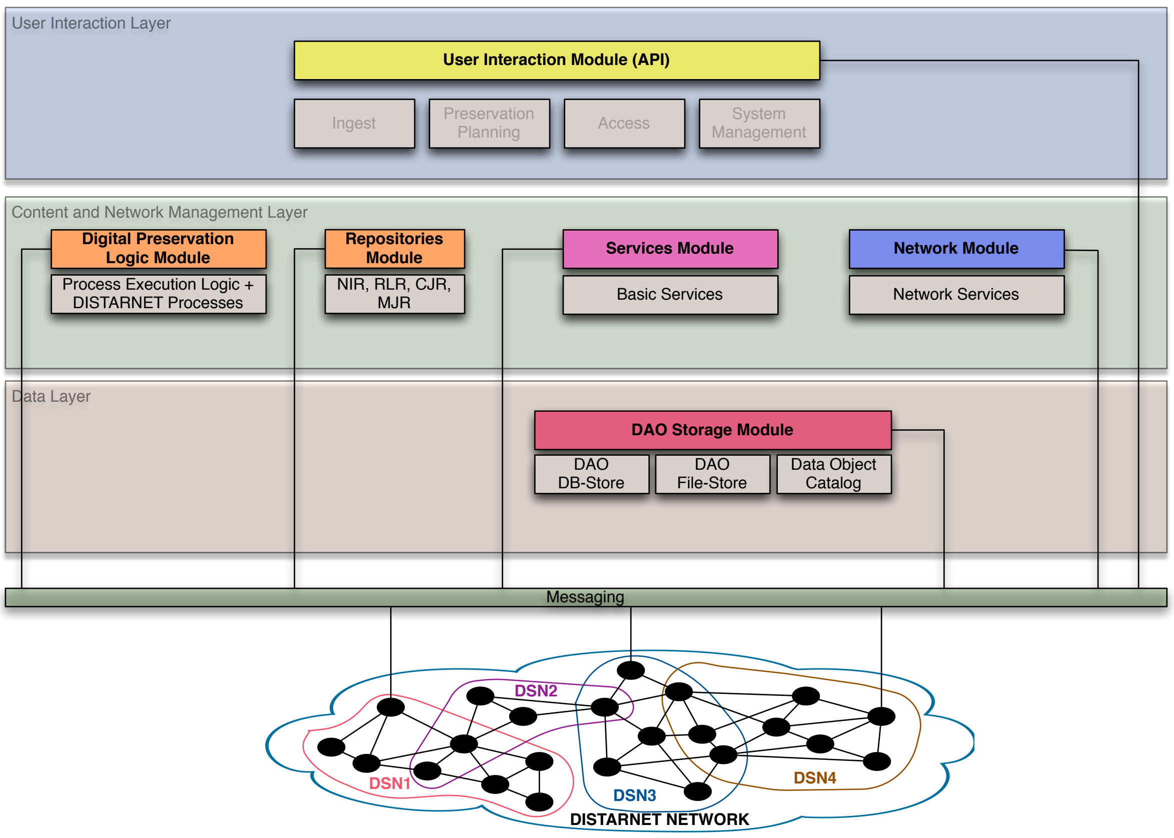

Figure 3.2: Modularized View of the System Model

We will start by presenting another view of the system model, in which the building blocks are organized into modules. Following this, we will present the DISTARNET processes and the DISTARNET model into greater detail.

The different building blocks in the three layers of the DISTARNET system model, which we have briefly described, are additionally organized into modules. Figure 3.2 depicts the modularized view of the system model.

The three layers are broken down into independent functional modules which communicate with each other by exchanging messages. The reasoning behind this modularization is threefold. First, the separation of concerns: We want to have building blocks with a clear separation of functionality, and with minimum overlap and dependencies, so that they can function independently of each other as much as possible. Secondly, stemming from the needs of the implementation side, this modular design provides the ability to exchange independently and/or extend each modules functionality with different implementations in the future. The only restriction imposed on alternative implementations is the correct processing of the predefined messages and the external effects, but they are otherwise free of any restrictions regarding the internal implementation. Thirdly, the modularization would provide additional possibilities on the implementation side, for fault-tolerant behavior on the node level.

The three DISTARNET node layers are subdivided into six functional modules. The *User Interaction Layer* is composed solely of the *User Interaction Module* which encompasses this layers functionality. The *Content and Network Management Layer* is divided into four modules which are the *Digital Preservation Logic Module*, the *Repositories Module*, the *Services Module*, and the *Network Module*. The third layer, the *Data Layer* is contained in the *DAO Storage Module*.

Module Manager and Submodules

To allow for maximum flexibility in the structure inside each module, every module is further divided into submodules, where each of them provide a certain part of the modules functionality. On top of all the submodules, there is also always a *Module Manager*, which functions as an access point to all communication with the module. Every message originating outside of a module, and is intended for a submodule, is sent to the modules manager and then further routed to the intended destination. Such a design provides a large flexibility as the “interface” seen from the outside of the module consists of the modules “address”, i.e., the manager, and the messages themselves. The structure inside each module can be changed, without having to perform any additional changes outside of the module.

3.4.1 User Interaction Module

The *User Interaction Module* consists of the UI Manager providing an API, and different submodules that by using the API provide the functionality to interact with the system on user and management levels. Figure 3.3 shows the structure of the User Interaction Module.

The API provides a standardized interface that allow a two-way communication with all other modules. The specification of the submodules providing Ingest, Preservation Planning, Access, and System Management functionality is out of the scope of

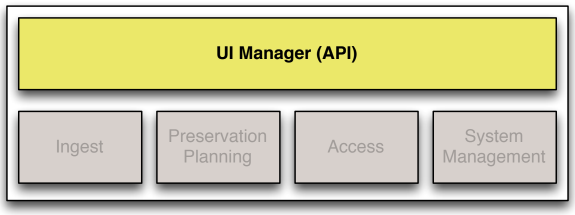

Figure 3.3: User Interaction Module

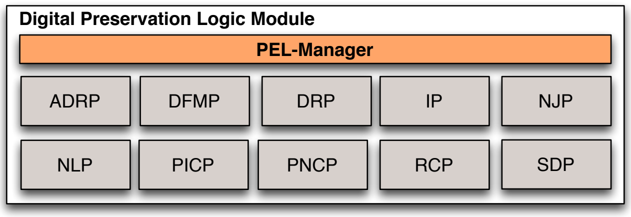

Figure 3.4: Digital Preservation Logic Module

our design.

3.4.2 Digital Preservation Logic Module

The Digital Preservation Logic Module is the central module, where the *Process Execution Logic (PEL)* and the *DISTARNET Processes* are housed. Figure 3.4 shows an overview of the Digital Preservation Logic Module. At the top, we have the PEL-Manager, which is the access point for this module, and is responsible for the routing of messages destined to any submodule. Further, the PEL-Manager incorporates also the Process Execution Logic, which is responsible for the execution, monitoring, and management of the DISTARNET processes. Beneath the PEL-Manager, we have the DISTARNET processes. In the following, we discuss the functionality of the Process Execution Logic, and who and when triggers the execution of the DISTARNET processes. The DISTARNET processes will be discussed in greater detail in Section 3.5.

Process Execution Logic

The *Process Execution Logic (PEL)* is the central controlling instance for the execution, management, and monitoring of all processes in the system. Each process execution is initiated by the Process Execution Logic. Further, every running process is monitored

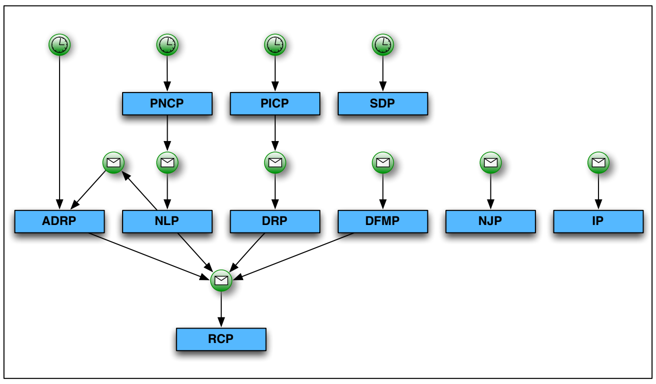

Figure 3.5: Process Execution Logic Trigger Hierarchy

by the PEL, and if required terminated and/or restart in case of failure.

The PEL will execute processes as a response to messages originating from either a timer, other processes, or by operators of the system. Figure 3.5 depicts the different process execution triggers for each process. Processes connected to a watch symbol are initiated periodically while the processes connected to a message symbol are initiated on demand by other processes, or directly by the system operator.

Periodic Neighbor-Node Checking Process (PNCP): The PNCP is triggered periodically, and checks if all members of the same DISTARNET Sub-Network (DSN) are still available. In case that a remote node is detected as lost, the PNCP will trigger the NLP.

Node Lost Process (NLP): The NLP is triggered by the PNCP when a node loss is detected. The NLP will trigger the RCP and the ADRP.

Automated Dynamic Replication Process (ADRP): The ADRP is either triggered periodically (e.g., at least once a day) or by the NLP for each DAO in the archive. In case that the degree of replication for a DAO is not sufficient, or the placement on remote nodes is not optimal, the ADRP will trigger the RCP for the DAO in question.

Periodic Integrity Checking Process (PICP): The PICP is triggered periodically (e.g., at least once a day) for all DAOs in the archive, and checks the integrity of all DAOs stored on a node. In case that a DAO corruption is detected, the PICP will trigger the

DRP.

DAO Repairing Process (DRP): The DRP is triggered by the PICP when DAO corruption is detected. It will take the needed measures to repair the DAO, and by doing so triggers the RCP.

Data Format Migration Process (DFMP): The DFMP is triggered on demand by the curator. This is not a daily maintenance job, and for the running of this process, additional information is needed, that first needs to be provided by the curator of the archive in the form of a migration job description.

Reliable Copying Process (RCP): The RCP is either triggered by the ADRP and DFMP in the outbound direction, or the NLP and DRP in the inbound direction, and will send DAOs to remote nodes (outbound), or get DAOs from remote nodes (inbound).

State Dissemination Process (SDP): The SDP is triggered periodically and disseminates data that was changed since the last time it was executed.

Node Joining Process (NJP): The NJP is triggered on demand by the administrator.

Ingest Process (IP): The IP is triggered on demand by the administrator.

General Process Failure Characteristics

All DISTARNET processes are of idempotent nature, and thus can be simply restarted in case of process execution failure. The PEL monitors the execution of all processes, and acts upon any failures encountered during process execution, which can not be handled by the processes themselves, by either simply terminating a process, or if needed by also restarting them.

Process Persistence

Every process managed by the PEL is persisted in such a way, that after a system shutdown the process execution of the processes running before the shutdown, can be resumed. The specifics regarding process persistence is implementation specific and depends on the implementation of the Process Execution Logic. We will discuss this further in the Section 4.4.3.

Process Versioning

A long-term preservation system, with long life expectancy, needs to have the ability of allowing processes to evolve, so that the system can be improved or adapted to changing requirements.

It will not be possible to update directly or change the already running processes, but new versions of processes can be deployed.

When deploying an updated process, we need to define what should happen to the already running process instances based on older versions of the process. There are three strategies that can be followed in this regard:

- *Proceed:* The old version process instances proceed their execution of the process as defined by the process definition at the time the process was started. As a consequence, these processes will resume their execution as if there never was an update of the process definition. Newly created instances can be started by using the new process definitions.
- *Abort and Restart:* The process instances based on the old version that are running will be aborted. If need, the process instance can be restarted by using the new process definition.
- *Migrate:* The running old version process instances are migrated in the middle of the process execution to the new process definition, and will run after a successful migration based on the updated process logic.

We follow the *Abort and Restart* strategy, where already running process instances will be restarted with the new process definition.

3.4.3 Repositories Module

This module includes the different repositories storing information needed for the execution of the DISTARNET processes. The *Node Information Repository - NIR*, the *Replica Location Repository - RLR*, the *Copy Job Repository - CJR*, and the *Migration Job Repository - MJR*. Figure 3.6 shows the structure of the Repositories Module. At the

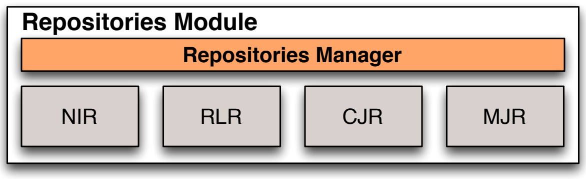

Figure 3.6: Repositories Module

top, we have the *Repositories Manager (RM)* responsible for the routing of all messages destined for the different repositories. The RM is also responsible for the monitoring of the different repositories, and in case of a fault not handled by the repositories, to try and restart the one in question.

3.4.3.1 Node Information Repository

The Node Information Repository (NIR) contains information about every *Node* in all the DISTARNET Sub-Networks (DSN) a node is part of. If a node is part of more than one DSN, it will simply have more than one entry. The attributes that each entry can hold are shown in Table 3.4. First we have the attributes identifying an entry, the unique identifier of the DSN and the node. Next are the entries from the Periodic Neighbor-Node Checking Process, the timestamps of when the last time checkNode messages where sent to this node, and when replies where received, and the status of the node. Further we have entries with location information of a node like the country code, or the latitude and longitude coordinates of a node. Finally, we also have information regarding available resources for this node (e.g., free/used space, etc.), and the timestamp of the last change of an entry. The entries in the NIR are distributed to other nodes through the State Dissemination Process.

3.4.3.2 Replica Location Repository

The Replica Location Repository (RLR) contains information about each *DAO replica* from every DISTARNET Sub-Network (DSN) where the node is a member of. Every replica entry is identified with the unique identifier of the DSN, the name of the node this replica is stored on, and the URI of the DAO this replica pertains to. Additionally an entry will contain information about the currently responsible node for the execution of the ADRP, and information about the node on which the DAO was originally

Attribute	Description
DSN	Unique identifier of the DSN
node	Unique identifier of the Node
lastCheckNodeSent	The timestamp of the last checkNode message sent
lastCheckNodeReceived	The timestamp of the last reply to a checkNode message
status	The state of the node (OK, LOST)
countryCode	The country code
geoLat	Latitude coordinate of the node
geoLong	Longitude coordinate of the node
freeSpace	Available space on this node reserved for the DSN
timestamp	Timestamp of entries last change

Table 3.4: Node Information Repository Entry Attributes

Attribute	Description
DSN	Unique identifier for the current DSN
node	Unique node identifier that holds a replica
uri	URI of the DAO for which this entry is meant
responsible	The node responsible for the execution of the ADRP
ingester	Unique node identifier of the node that ingested this DAO
status	The state of the replica on this node (OK, CORRUPT, LOST)
timestamp	Timestamp of the last change

Table 3.5: Replica Location Repository Entry Attributes

ingested. Table 3.5 lists all the attributes that an entry will contain.

Processes that create or change entries are the Ingest Process, Periodic Integrity Checking Process, the Remote Copying Process (i.e., RCPOutRemote), and the Periodic Neighbor-Node Checking Process. The entries in the RLR are distributed to other nodes through the State Dissemination Process.

3.4.3.3 Copy Job Repository

The Copy Job Repository (CJR) is used to store information about copy jobs that need to be executed by the Remote Copying Process. An entry is uniquely identified by the unique identifier of the DISTARNET Sub-Network (DSN), the unique node identifier

Attribute	Description
DSN	Unique identifier for the current DSN
node	Unique node identifier of the node executing the job
uri	URI of the DAO for which this entry is meant
locations	Locations of the receiving end, or the locations of the source end
parts	The parts of a DAO that need to be copied. Empty if the whole DAO is to be copied.
direction	The direction of the copy job (inbound, outbound)
type	The process type that created the copy job (outbound: ADRP, DFMP; inbound: DRP, NLP)
status	The status of the job (open, closed, failure message)

Table 3.6: Copy Job Repository Entry Attributes

of the node executing the job, and the URI of the DAO which is to be copied. Next, each entry will hold either the locations of nodes that will receive the DAO in the case of an outbound job, or the locations of the source nodes from which the DAO can be copied in the case of an inbound job. If only parts of an DAO are to be transported, then a list of those parts can be stated. Also, every entry will hold the type of the process that created the entry. Possible types are from outbound jobs ADRP and DFMP, and for inbound jobs DRP and NLP. Finally, the status of each job is stored with each entry being op, closed, or in case of a failure the message text. All attributes of an entry in the Copy Job Repository are listed in Table 3.6.

3.4.3.4 Migration Job Repository

The Migration Job Repository (MJR) store information needed for the execution of the Data Format Migration Process (DFMP). An entry in the MJR is identified by the unique identifier of the current DISTARNET Sub-Network (DSN), the unique node identifier of the node executing the job, and the URI of the DAO that needs to be migrated. Additionally, each entry holds the path to the migration script that will be executed by the DFMP, and status of the job being open, closed, or a failure message,

Attribute	Description
DSN	Unique identifier for the current DSN
node	Unique node identifier of the node executing the job
uri	URI of the DAO for which this entry is meant
path	The node responsible for the execution of the ADRP
status	The status of the job (open, closed, failure message)

Table 3.7: Migration Job Repository Entry Attributes

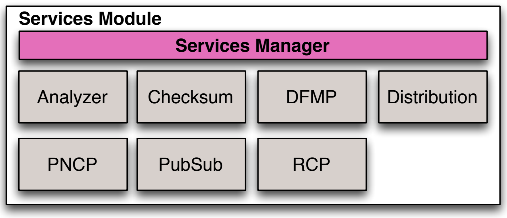

Figure 3.7: Services Module

depending on the result of the process execution. Table 3.7 shows the attributes of the entries that will be stored in the Migration Job Repository.

3.4.4 Services Module

The *Services Module (SM)* houses the basic-services which are used by the processes in the Digital Preservation Logic Module. Figure 3.7 shows an overview of the Services Module. At the top of the SM, we have the *Services Manager*, and underneath the different submodules containing the basic-services.

Again, the Services Manager is the entry point for any communication with the Services Module, and all messages are routed to their destination. The Services Manager is also responsible for the monitoring of the submodules, and in case of any faults not handled by the submodules, to try and restart them.

3.4.4.1 Analyzer Basic-Services

The *Analyzer* submodule contains the *AssertDAODamage* basic-service, which can be used for the assertion of the damage of a DAO. This service is used by the DAO Repairing Process, to analyze the DAO, and as a result, return those parts of the DAO, which need repair.

3.4.4.2 Checksum Basic-Services

The *Checksum* submodule contains two basic-services, the *ExtractDAOChecksums*, and *CalcDAOChecksums*. The ExtractDAOChecksums service can be used to extract existing checksums stored as part of the DAO. The CalcDAOChecksums service can be used to recalculate checksums of a DAO.

3.4.4.3 DFMP Basic-Services

The *DFMP* submodule contains one basic-service, which can be used for execution of data format migration jobs.

3.4.4.4 Distribution Basic-Services

The *Distribution* submodule consist of two basic-services, namely the *CalcOptimalNrOfReplicas* used to calculate the optimal number of replicas, and the *CalcOptimalDistribution* used to calculate the optimal replica locations.

Estimating the Optimal Number of Replicas

The CalcOptimalNrOfReplicas service will estimate the optimal number of replicas needed by taking into account the availability of nodes (based on statistics on the individual availability collected in the past) used to store a data object in conjunction with the availability threshold imposed by the preservation policy of the data object itself. This estimate will be used to raise or lower the number of replicas if needed.

In DISTARNET the availability of a data object depends on the failure rate of the nodes. In the case that a larger number of nodes become unreachable can lead to the unavailability of a data objects. The following formula will estimate the needed number of replicas (n) for a certain availability threshold.

Let n be the total number of replicas for a data object, p the average probability of a site to be up and a_d the required amount of availability for a data object D:

$a_d = 1 - (1 - p)^n$

Thus if we take 90% average probability of a site to be up (90% availability of a node equals to 36.5 days of downtime per year) and if we have at least 3 replicas in the network, then the availability of the data object will be 99.9% which equals to 8.76 hours per year the data object will not be available.

So taking the needed average availability of the archived data objects and the average availability of the nodes used to store them into account, the system can calculate the optimal number of replicas needed.

Calculating the Optimal Replication Locations

The CalcOptimalDistribution service will try to use the largest available granularity for the distribution of the replicas, i.e., the information about collections/subcollections will be used, if available. This means that the replicas of data objects belonging to a collection/subcollection will be stored together if possible.

Using data from the local *Node Information Repository*, a ranked list of optimal nodes is calculated. This list contains the most suitable geographically dispersed nodes for storing the replicas by taking into account the available storage resources and any preservation policy-based restrictions.

3.4.4.5 PNCP Basic-Services

The *PNCP* submodule contains one basic-service used to send and receive messages to and from other nodes with the purpose to check if the nodes in the same DISTARNET Sub-Network are still alive and responding. Upon receiving reply messages, the service will additionally update the entry for the sending node in the Node Information Repository with the timestamp when the reply was received.

3.4.4.6 PubSub Basic-Services

The PupSub submodule contains the basic-services used to send and receive the state of a node with the publish-subscribe pattern [83]. On the sending side, the service takes as the input values the states that need to be send, and the name of the DISTARNET Sub-Network. The service will then send the state information to all nodes belonging to the DSN. On the receiving side, the service receives states sent from other nodes, and stores them to the appropriate repositories (Node Information Repository or Replica Location Repository).

3.4.4.7 RCP Basic-Services

The RCP submodule contains basic-services used by the Remote Copying Process. These services can be used to initiate secure transfer of DAOs (or parts of DAOs) to

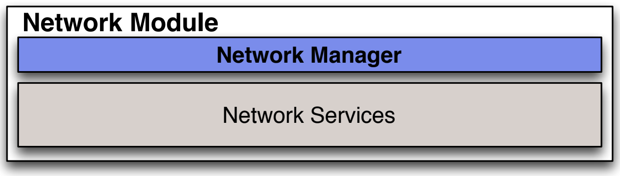

Figure 3.8: Network Module

remote nodes (outbound), and to initiate secure transfer of DAOs from remote nodes to the local node (inbound). The transferred payload is rechecked whether it really has been successfully transferred, and no data has been lost or written inconsistently during the copy process. For consistency checking, the service calculates checksums (e.g., SHA-1) on both ends (before and after the transfer) and compares them to each other. If any inconsistencies are found, then the transfer is repeated.

3.4.5 Network Module

Figure 3.8 shows the overview of the Network Module, where at the top we have the *Network Manager* responsible for the routing of messages to the appropriate submodule.

3.4.5.1 Network Services

The Network Services submodule is responsible for the routing of messages to remote nodes. It implements a simple name to full address resolver, where messages internally addressed with DISTARNET Sub-Network (DSN) Name and Node Name, are looked up in the DSN Member Registry, and forwarded accordingly. On the other side, this sub-module receives messages from other remote nodes and routes them to the intended recipient.

3.4.6 DAO Storage Module

Figure 3.9 shows an overview of the *DAO Storage Module*. At the top of the Storage Module, we have the *Storage Manager* acting as the entry point for any communication with the module, which is responsible for routing any messages to the intended recipient submodule. The Storage Manager is also responsible for restarting of the

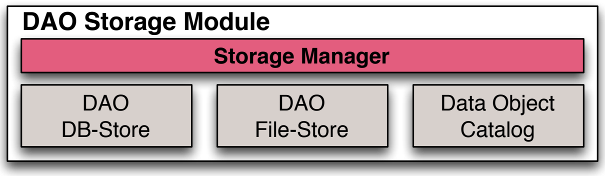

Figure 3.9: DAO Storage Module

submodules in case of failure.

Further, we have the *DAO DB-Store* a database-based storage for the DAO metadata, the DAO File-Store representing a filesystem-based storage for DAO metadata, and the Data Object Catalog being the storage facility for the bistream referenced by the DAO metadata. The DAO DB-Store together with the Data Object Catalog is responsible for the storage of the DAOs. The DB-Store stores the metadata part of the DAO and the Data Object Catalog the bistream part. The DAO File-Store stores the data in a different representation and just mirrors the DAO DB-Store.

A detailed discussion regarding the structure of the DISTARNET Archival Object which is stored in the DAO Storage Module is provided in Section 3.6.

3.4.6.1 DAO DB-Store

The *DAO DB-Store* is responsible for the database-based storage and management of DAO metadata. This submodule contains basic-services for accessing to the DAOs stored in the DB, providing transactional read, write, and update functionality.

3.4.6.2 DAO File-Store

The data stored in the *DAO File-Store* represents the same data as stored in the DAO DB-Store, where the only difference lies in the serialization format of the data. This redundant storage has a twofold motivation. The explicit representation stored on the filesystem provides the basis for exporting, transporting, and archiving of the DAOs with their asserted relationships to other objects. The database-based store provides an index of an entire repository and the basis for high-performance queries over their relationships. An added advantage of the dual representation is that the entire database-based store can be rebuild by importing and parsing file-based DAO representations.

3.4.6.3 Data Object Catalog

The *Data Object Catalog* is responsible for the storage of the bitstream data belonging to DAOs. The basic-services contained in this submodule provide the functionality needed for read and write access to the bitstream data.

3.5 DISTARNET Processes

In the previous section, we have discussed the different modules, and the different functionalities that each module provide. While discussing the *Digital Preservation Logic Module*, we have only given a short overview of the different DISTARNET processes defined and executed inside this module.

In the following, we take a look at each DISTARNET process individually. We discuss the individual steps that each process is made of and how it all plays together with the other modules, which provide the different services that are consumed during the execution of the processes.

3.5.1 Ingest Process

The *Ingest Process (IP)* is the process which is used to add the DAOs into the archiving environment. During the IP the different parts of a DAO are stored into the Storage Module. The data representing the metadata of the DAO, which is basically everything besides the bitstream, is stored into the DAO DB-Store and the DAO File-Store. The remaining, bitstream data, is stored in the Data Object Catalog.

In the course of the IP, an entry will be added to the Replica Location Repository for every DAO that is ingested. Each entry contains the location (i.e., DSN, current node), the URI, the responsible node (current node), the ingesting node (current node), status, timestamp. The entries in the Replica Location Repository are distributed to the other nodes in the DSN through the State Dissemination Process (SDP).

3.5.2 Node Joining Process

The *Node Joining Process (NJP)* is the process in which a node joining the network is authorized to participate in the DISTARNET network. Depending on the type of the DiSTARNET network this process can have a range of different behaviors:

Public DISTARNET network: In the case of a public network, the joining node is supplied with a number of seed nodes. These seed nodes can than be queried for information about other nodes in the network, and so on, until the newly joined node has information about all nodes in the public DISTARNET network, which is then stored in the Node Information Repository.

Small private DISTARNET network: The joining node is provided with basic information about all other participants, which is stored in the Node Information Repository. All the other participants also add the newly joining node to their list of authorized members, and the Node Information Repository.

Large private DISTARNET network: The joining node is provided with credentials that will allow it to authorize itself to other members in the network. The node will also be provided with a number of seed nodes that it can query for information about other nodes in the network. After successful mutual authorization of the new joining node with all other nodes in the network, they will all add each other to their Node Information Repository.

3.5.3 Periodic Neighbor-Node Checking Process

The *Periodic Neighbor-Node Checking Process (PNCP)* is responsible for initiating periodic communication, with every node in the *DISTARNET Sub-Network (DSN)* the node is a member of, to check if they are alive. If a node that is being checked is failing to respond for a predefined amount of times, then a *Node-Lost Event* is generated, which subsequently triggers the *Node-Lost Process (NLP)*.

The PNCP is depicted in Figure 3.10. The process is started when the PNCPStart message is received with the DSN as the parameter. It begins by querying the Node Information Repository (NIR) for the members of the DSN (P01-R01), and then by sending check messages to all nodes by using the PNCP basic-service (P02-S01). After the messages are sent, a timestamp is written to the NIR (P03-R03) for every receiving node. Asynchronously, the PNCP service receives replies from the sent messages and writes their timestamp to the NIR for every sending node (S01-R03). Following the process again, the NIR is queried for the last reply times of every node (P03-R04). If the last reply time is older then 24h, then the node is marked as lost in the NIR, and a *Node-Lost Event* is triggered (P05-P06). If the last reply time is not older than 24h, then the process simply ends.

The process was designed with asynchronicity in mind, i.e., for the process execution not to have any dependencies on the PNCP service for the received reply messages. The process can continue its execution even if the remote nodes are not responding immediately. This process can be terminated and restarted at any time without the need to store or restore any state information.

3.5.4 Node-Lost Process

The Node-Lost Event that was triggered in the PNCP will subsequently trigger the running of the *Node-Lost Process (NLP)* depicted in Figure 3.11.

The process is started with the NLPStart message, and the DSN and name of the node which is lost as the parameter. The process begins by marking all entries in the Replica Location Repository (RLR) with the status *LOST* (P01-R01). Afterwards, the process checks if the current node has the responsibility for the DAOs ingested on the lost node, i.e., is the next node in the responsibility chain (P02-R02). If the node is not responsible for the lost node's DAOs, then the Automated Dynamic Replication Process (ADRP) is called (P11) so that additional copies for the lost ones can be created. If the node, on the other hand, is responsible for the lost node's DAOs, then the current node takes over the responsibility for these DAOs by updating the RLR entries for the DAOs ingested on the lost node, and adding the current node as the responsible node (P03-R03). As the next step, the process checks if the DAOs for which the node is now responsible are already stored locally (P03-R04). If they are all stored locally already, then ADRP is called, which will now run for both the DAOs ingested on the local node, and for the foreign DAOs the node is responsible now (P11). In the case that some DAOs are not stored locally (P04), the process retrieves the remote locations where the replicas of the DAOs are stored (P05-R05), creates inbound copy jobs for the DAOs that we want to store locally (P06-R06), and calls the Remote Copying Process (RCP) (P07). After this, the first execution strain of the process ends.

The retrieval of the DAOs that we want to store locally happens asynchronously. As soon as RCP has retrieved a DAO, another instance of the NLP is started with the RetrieveDAOResult message, which starts the second execution strain of the process. This strain is executed for every DAO for which we have created a copy job in the first part of the process. The execution starts with the evaluation of the retrieval result. In

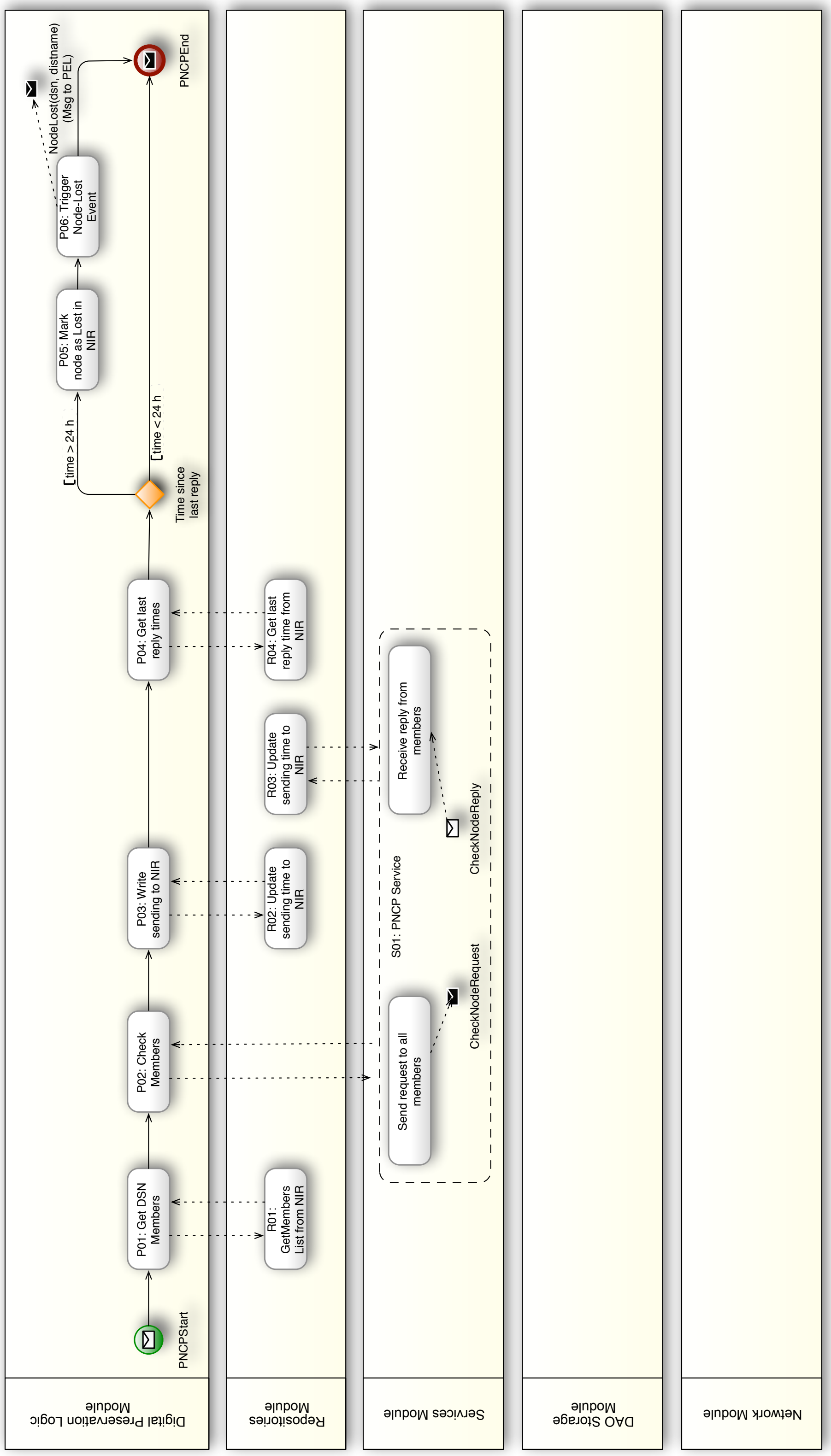
Digital Preservation Logic Module
Repositories Module
Services Module
DAO Storage Module
Network Module
PNCPStart
P01: Get DSN Members
R01: GetMembers List from NIR
P02: Check Members
Send request to all members
CheckNodeRequest
S01: PNCP Service
CheckNodeReply
Receive reply from members
P03: Write sending to NIR
R02: Update sending time to NIR
R03: Update sending time to NIR
P04: Get last reply times
R04: Get last reply time from NIR
Time since last reply
[time > 24 h
[time < 24 h
P05: Mark node as Lost in NIR
P06: Trigger Node-Lost Event
NodeLost(dsn, distname) (Msg to PEL)
PNCPEnd

Figure 3.10: Periodic Neighbor-Node Checking Process (PNCP)

the case of success, the DAO is stored locally (P08-ST01), and the RLR is updated to reflect the location of the new replica (P09-R08). Next, the RLR is queried if now all DAOs for which we are responsible are stored locally (P10-R09). If they are not, then the process simply ends and if they are all retrieved now, then ADRP is called, which will as before run for both the DAOs ingested on the local node, and for the foreign DAOs the node is responsible now (P11). For the case that the DAO retrieval has failed, then the RCP job will be updated in such a way that the retry count is incremented, the job status changed to OPEN (P12-R07), and RCP called again (P13).

The reasoning behind the two independent execution strains is the resulting advantage, as the lower part of the process from P01 until P07 is executed only once, and the upper part of the process is executed for every retrieved DAO. With this design, we have many short running instances of the process, and after the lower part has run, or a retrieved DAO was processed, we do not need to keep the process running anymore, or track their state. An alternative design would have been to build one long-running process, but this would have been more costly as the Process Execution Logic would need to keep track of the running process, and route the RetrieveDAOResult messages to the right process instance.

3.5.5 Automated Dynamic Replication Process

The *Automated Dynamic Replication Process (ADRP)* as depicted in Figure 3.12, is responsible for finding suitable storage nodes for remote replicas, estimating the optimal number of replicas, and initiating the creation of new replicas.

The process begins after receiving the ADRPStart message containing as a parameter the name of the DSN and the URI of the DAO that we are evaluation. As the first task, the process retrieves the preservation policy of the DAO from the storage module (P01-ST01). Next the optimal number of replicas is calculated (P02-S01), the current storage locations from the RLR (P03-R01), and the members of the DSN from the NIR (P04-R02) are retrieved. These information are than used to calculate the optimal distribution for the DAO (P05-S02). Afterwards, the calculated distribution is compared with the current distribution, and a decision is made if additional replicas need to be created. If the distribution is found not to be optimal, then an outbound copy job is created the Reliable Copying Process called (P06-R03, P07).

We saw earlier that a node can be a member of one or more DSNs. The ADR process

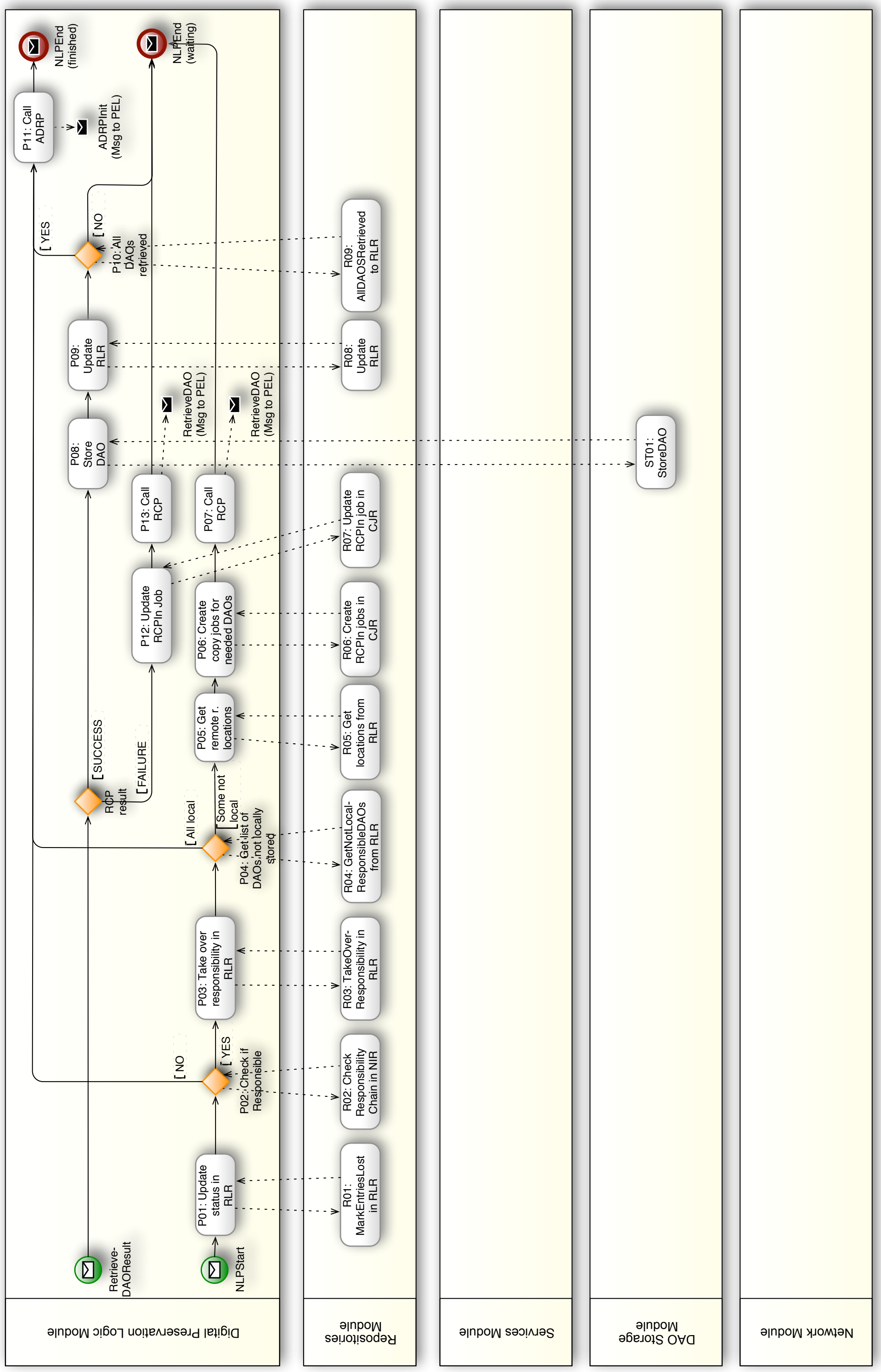
Digital Preservation Logic Module
Repositories Module
Services Module
DAO Storage Module
Network Module
NLPStart
Retrieve-DAOResult
P01: Update status in RLR
P02: Check if Responsible
[YES
[NO
P03: Take over responsibility in RLR
P04: Get list of DAOs not locally stored
[All local
[Some not local
P05: Get remote r. locations
P06: Create copy jobs for needed DAOs
P07: Call RCP
RetrieveDAO (Msg to PEL)
RCP result
[SUCCESS
[FAILURE
P12: Update RCPIn Job
P13: Call RCP
RetrieveDAO (Msg to PEL)
P08: Store DAO
P09: Update RLR
P10: All DAOs retrieved
[YES
[NO
P11: Call ADRP
ADRPInit (Msg to PEL)
NLPEnd (finished)
NLPEnd (waiting)
R01: MarkEntriesLost in RLR
R02: Check Responsibility Chain in NIR
R03: TakeOver-Responsibility in RLR
R04: GetNotLocal-ResponsibleDAOs from RLR
R05: Get locations from RLR
R06: Create RCPIn jobs in CJR
R07: Update RCPIn job in CJR
R08: Update RLR
R09: AllDAOSRetrieved to RLR
ST01: StoreDAO

Figure 3.11: Node-Lost Process (NLP)

will only operate within those boundaries, and use the storage resources of the DSN a DAO it is a member of.

Responsibility for ADRP Execution

Every DAOs redundancy and storage location is reevaluated periodically. This is done on the node where the DAOs where initially ingested, and only for those DAOs. The remote replicas of those DAOs are thus actually never directly reevaluated. The information regarding the ingesting node is stored in the RLR. Should an institution have more nodes but the data is ingested only at one node, then it will be possible to distribute the task of reevaluation to the other nodes by overriding the information regarding the ingesting node. The node who is initially set as the ingesting node is also at the same time set as the responsible node in the RLR entry.

In the case that the responsible reevaluation node is not available anymore (e.g., in the case of destruction), another node in the DISTARNET network will automatically take over. As soon as the original responsible node (i.e., the ingesting node) is available again it will contact the nodes storing the replicas – by using the information stored in the Replica Location Repository (RLR) – and inform them of his return and taking back the responsibility for triggering the ADR process.

The responsibility succession is set through the *responsibility chain*. The responsibility chain is a DSN wide ordered list, containing all nodes. In the case that one node is not available anymore, the next node on the list takes over the responsibility for the execution of the ADRP for DAOs ingested by the unavailable node. In the case that this node also becomes unavailable, then the next node on the list would take responsibility for the DAOs of both unavailable nodes, and so on.

Additional Optimizations

Additionally, to optimize the access performance the system will create if necessary additional replicas by analyzing the usage patterns of the digital objects.

3.5.6 Periodic Integrity Checking Process

Every node in a DSN will periodically (e.g., once a day) check the integrity of all locally stored DAOs using the *Periodic Integrity Checking Process (PICP)* depicted in

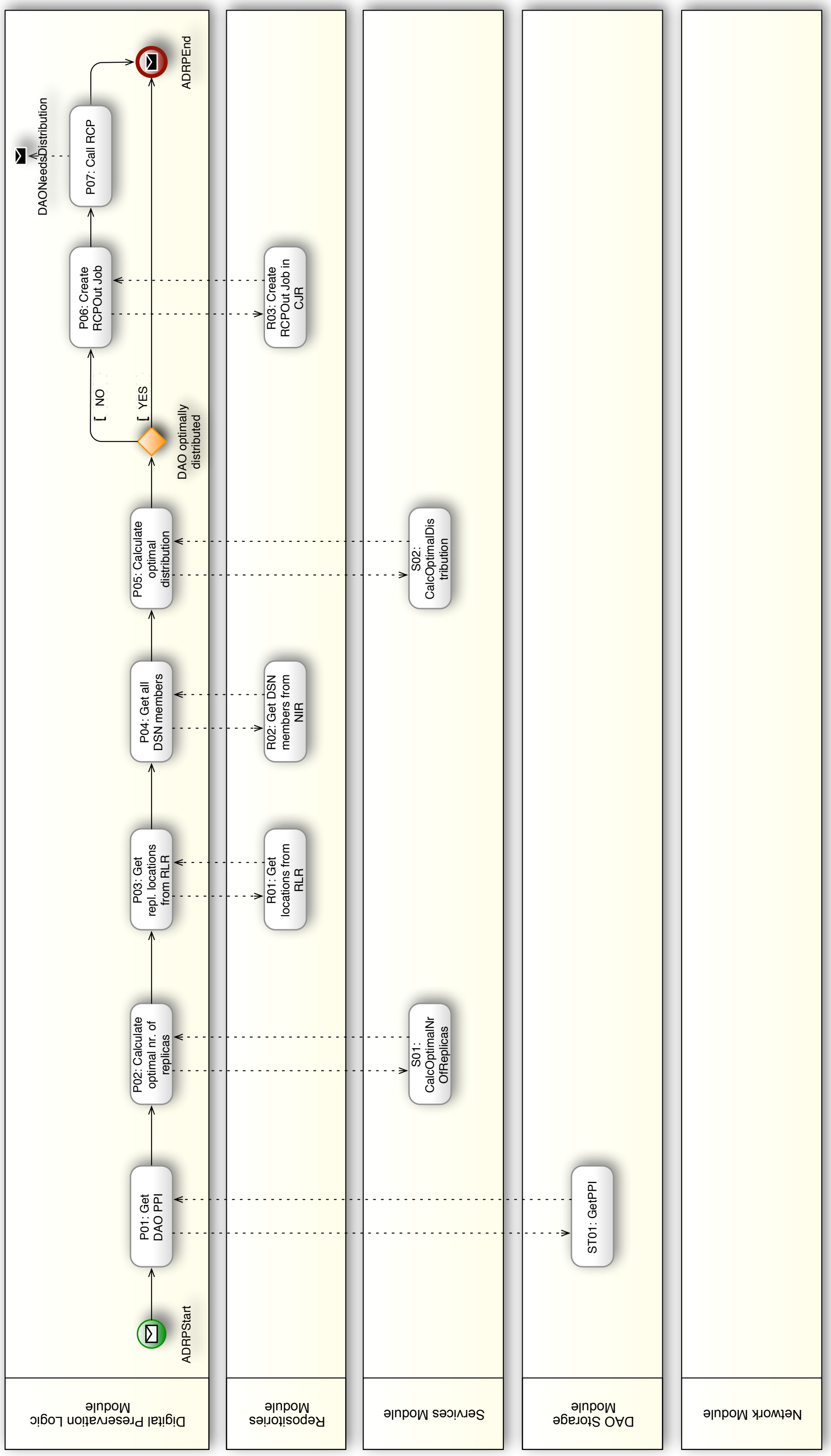
Digital Preservation Logic Module
Repositories Module
Services Module
DAO Storage Module
Network Module
ADRPStart
P01: Get DAO PPI
P02: Calculate optimal nr. of replicas
P03: Get repl. locations from RLR
P04: Get all DSN members
P05: Calculate optimal distribution
DAO optimally distributed
[NO]
[YES]
P06: Create RCPOut Job
P07: Call RCP
DAONeedsDistribution
ADRPEnd
R01: Get locations from RLR
R02: Get DSN members from NIR
R03: Create RCPOut Job in CJR
S01: CalcOptimalNrOfReplicas
S02: CalcOptimalDistribution
ST01: GetPPI

Figure 3.12: Automated Dynamic Replication Processes (ADRP)

Figure 3.13. After being successfully checked, a DAO is marked with a time-stamp. To eliminate overlap and excess integrity checking, only DAOs with a time-stamp older then X hours will be rechecked, where X is a network wide setting.

The PICP starts with the PICPStart message, containing the DSN and URI of the DAO. As a first step, the process retrieves the DAO from the DAO Storage Module (P01-ST01). Afterwards, it extracts (P02-S01), calculates (P03-S02), compares the extracted against the calculated checksum (P04). In the case that the checksums are equal, then the DAO is marked as checked (P05-R01). In the other case, where the extracted, and calculated checksums are not equal, the DAO is marked as corrupt (P06-R02), and the DAO Repairing Process (DRP) is called by sending the DAOCorrupt message to the PEL (P07).

Integrity

We check the integrity of a DAO by calculating and comparing checksums. During integrity checking, all parts of the DAO are taken into account, and checksums are calculated for the metadata and bitstream data parts of the DAO.

3.5.7 DAO Repairing Process

When the PICP finds that a DAO is corrupt, then it sends the DAOCorrupt message to the PEL. This message then triggers the initiation of the *DAO Repairing Process (DRP)* as depicted in Figure 3.14.

The process is started with the DRPStart message, containing the DSN and URI of the DAO. Next, the process retrieves the DAO from the DAO Storage Module (P01-ST01). Afterwards, the damage of the DAO is asserted by using the AssertDAODamage service (P02-S01). To be able to repair the DAO, we need to retrieve healthy parts from a remote copy. So next the process queries the RLR for remote replica locations (P03-R01), creates an inbound copy job (P04-R02), and sends a message to the PEL (P05) to initiate the retrieval of the needed parts. This part of the DRP ends here.

When the Remote Copying Process finishes the retrieval of the remote parts, it will inform the PEL which will in turn initiate the DRP, and start the process with the RetrieveDAOPartsResult message. Next, the returned result is analyzed, and if the retrieval was successful, this message will contain the needed parts for the repairing of the DAO. This is used in the following step, and sent to the DAO Storage Module to

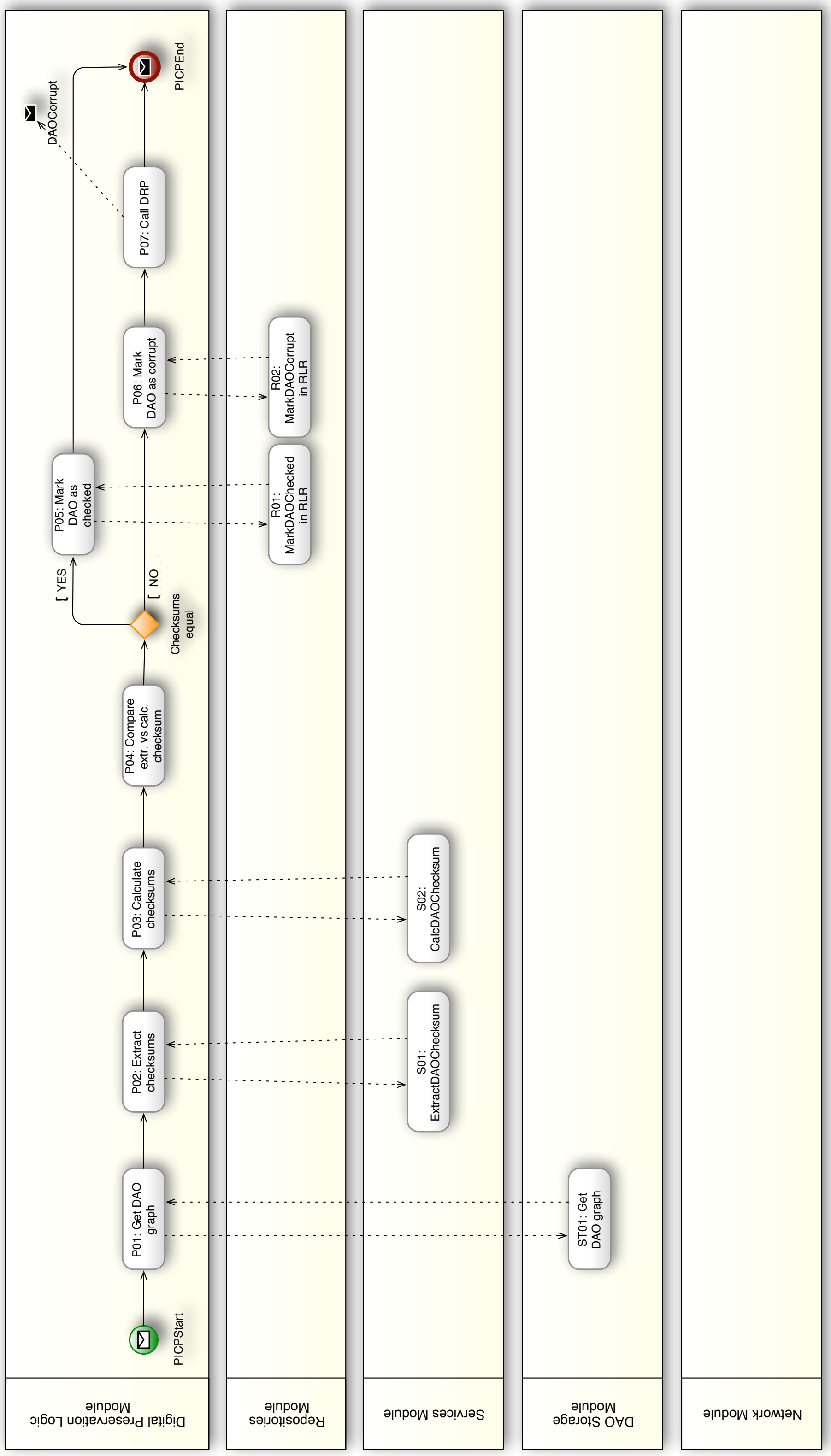

Figure 3.13: Periodic Integrity Checking Processes (PICP)

store the fresh data (P06-ST06). Afterwards, the DAO is marked as checked (P07-R03), and the process ends. In the other case, where the returned RCP result is negative, the DRP is called again.

As before in the NLP, this process has also two execution strains. The reasoning behind such a design is similar, in that we want short running process instances, and lower administration cost as we do not need to keep track of the process, after the first strain has finished. We can simply instantiate a new process which will start on the second strain.

3.5.8 Data Format Migration Process

The *Data Format Migration Process (DFMP)* is used for the on-demand migration of the archived content, i.e., the different representations constituting a DAO. For example, an image DAO, an object representing an image with different representations contains a representation in the TIFF format. The DFMP can be used to create an additional representation of the TIFF bitstream data by converting it to the JPEG2000 format, and afterwards appending it to the DAO.

As there is no limitation by the data model regarding the data format of the archival content, the process itself must be data format agnostic, and only provide a structure that allows the execution of migration scripts. This kind of flexibility is also required through the long-term nature of the archiving task as such, where the needs regarding data format migration in the distant future are not known at the present time.

Figure 3.15 shows the structure of the process, which begins by receiving the DFMP-Start message containing the DSN, URI of the DAO, and the job description, which we want to apply to the DAO. As a first step, the process calls the DFMP basic-service which executes the migration script (P01-S01). The process is then informed about the result, and in the case of an execution failure, the migration job is closed with the result (P06). If the execution of the migration job succeeds, then the result is appended to the DAO (P02-ST01). Afterwards, the so amended DAO needs to be transferred to remote nodes where a replica is stored, i.e., the remote replicas need to be also amended. For this, the process first retrieves the remote locations of any replica from the RLR (P03-R01), creates a copy job in which we only want to transfer the changed parts of the DAO (P04-R02), and finally trigger RCP (P05). At this point, the process ends.

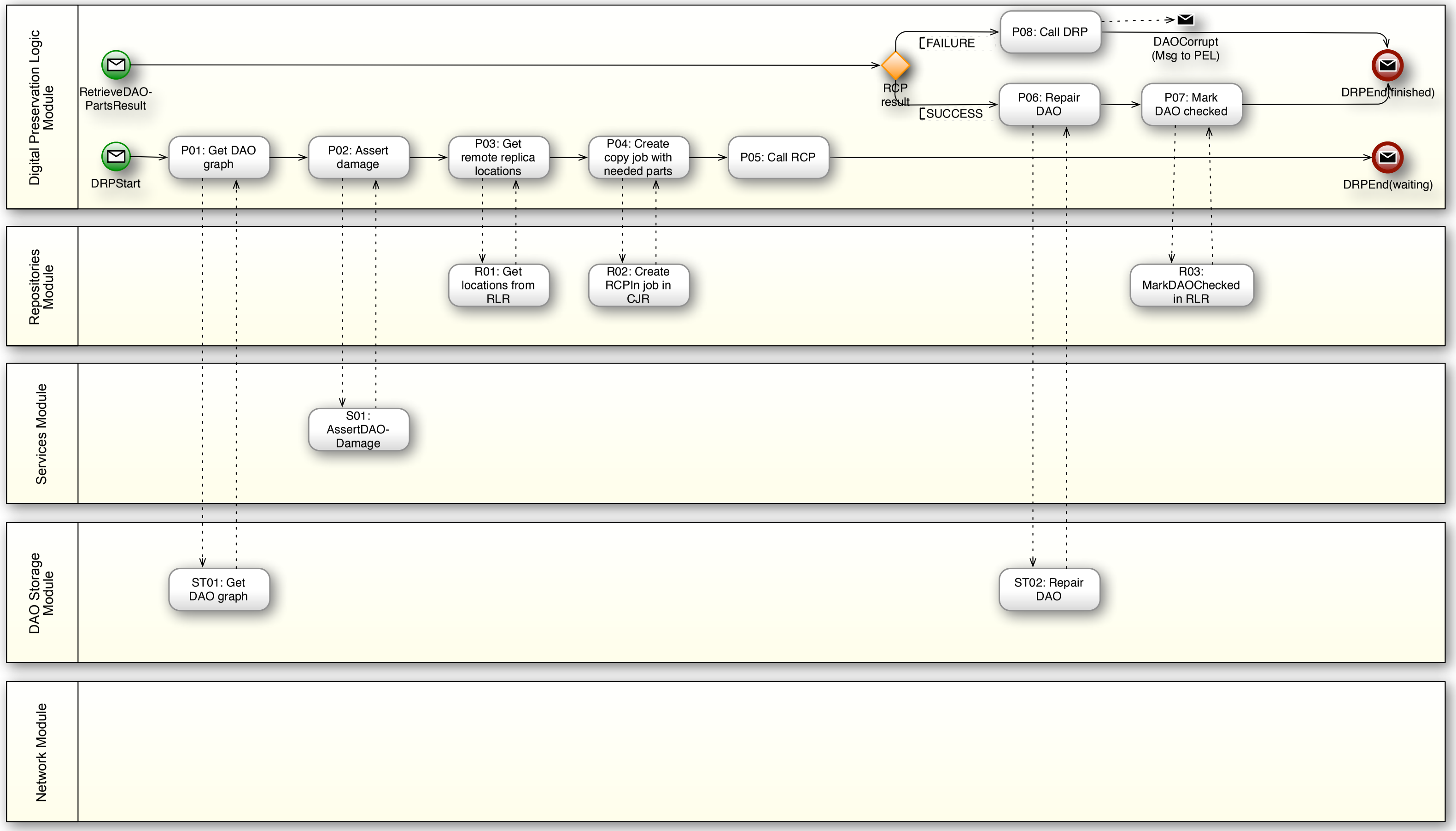

Figure 3.14: DAO Repairing Process (DRP)

The process is again started when the RCP job finished executing, and the result is sent with the DFMPRCPResult message, after which the migration job is closed with the corresponding result (P06).

3.5.9 Reliable Copying Process

The *Reliable Copying Process (RCP)* is responsible for creating new replicas of DAOs on remote nodes, or retrieving existing replicas from remote nodes, in a secure and efficient manner. At process level, transactional semantics according to the model of transactional processes [82] will be applied. It corresponds to the traditional data-carrier migration of digital data. This means that every copy has to be rechecked whether it really has been successful, and no data has been lost or written inconsistently during the copy process. This is achieved by calculating and comparing of checksums on the data streams transported between the nodes.

We differentiate between *outbound* and *inbound transfer* direction. The outbound transfer direction is used when new replicas of DAOs need to be created on remote nodes, and the inbound transfer direction is used when we want to retrieve existing replicas of DAOs from remote nodes to the local node. Figure 3.16 depicts the two transfer directions, and the processes involved in each. The *RCP-Out* and *RCP-Out-Remote* are used during outbound transfer, where as the *RCP-In* is used for inbound transfer.

For both the outbound and the inbound transfer direction, the content which is to be transferred, is defined in the *copy job description*, and is stored in the Copy Job Repository (CJR).

Outbound Transfer

The outbound transfer direction is used when creating new replicas of DAOs on remote nodes. For this, two processes are used, namely the RCP-Out process running on the local node who initiates the transfer, and the RCP-Out-Remote companion process running on the remote node. Figure 3.17 and 3.18 depict both processes.

An outbound transfer starts when the RCP-Out process receives the RCPOutStart message. The start message carries the DSN and the URI of the DAO that we want to transfer. As a next step, the process retrieves the copy job from the CJR (P01-R01),

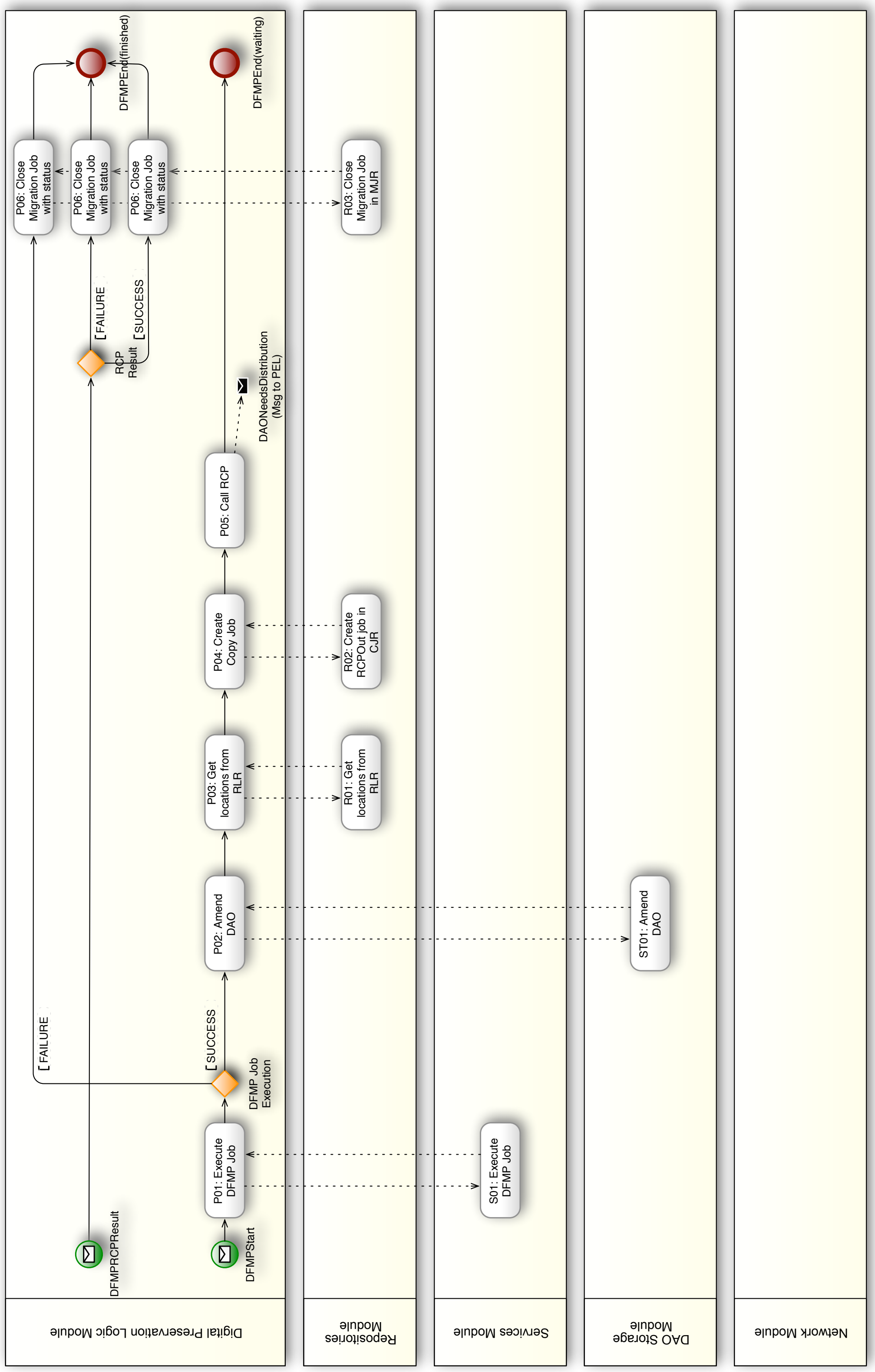

Figure 3.15: Data Format Migration Process (DFMP)

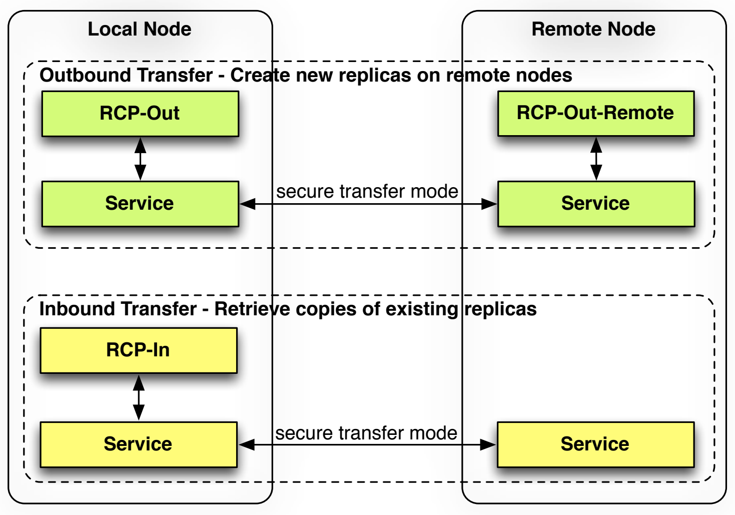

Figure 3.16: Reliable Copy Process Overview

and calls the RCPOut service with the parameters from the copy job containing destination information. The RCP-Out basic-service then retrieves the DAO from the DAO Storage Module (S01), and initiates the RCP-Out-Remote process on the remote node (S02). On the remote node, a RCP-Out-Remote process is started with RCPOutRemoteStart message. The message carries information allowing the RCP-Out-Remote basic-service started by the process (P01), to contact the RCP-Out basic-service on the initiating node (S01). After the DAO was received on the remote node (S02), the RCP-Out-Remote process is informed via the DAOReceived message. Afterwards, the DAO is stored, and depending on the outcome, the RLR is updated with information regarding the added DAO. In both cases, the initiating RCP-Out basic-service is informed about the outcome (P04 or P05). Now back on the initiating node, the RCP-Out basic-service receives the messages with the outcome (S04) and informs the RCP-Out process. The process closes the copy job with the result (P03-R02). Afterwards, depending on success or failure, and the process that created the copy job we send different messages (P04 or P05).

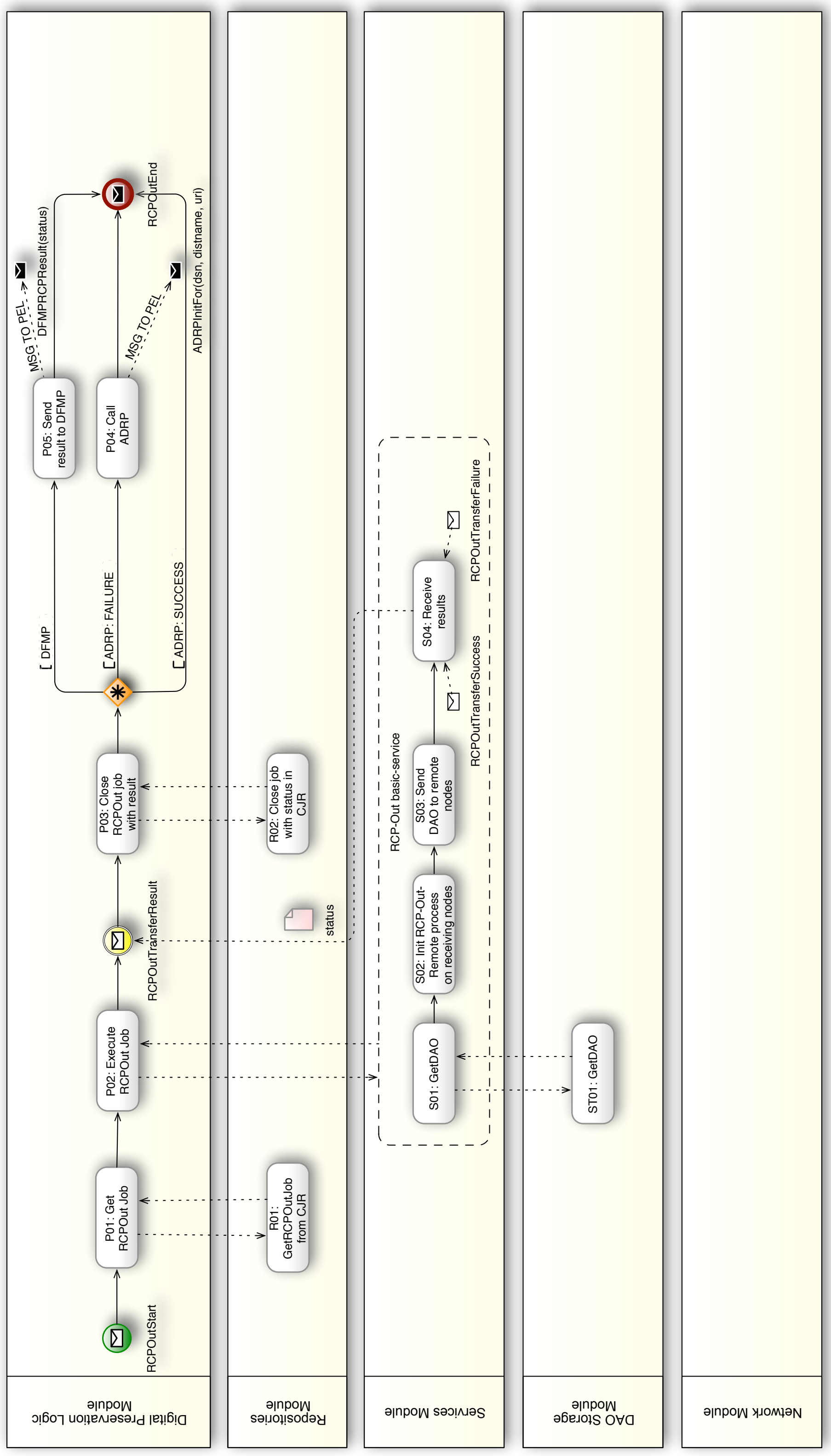

Figure 3.17: Reliable Copy Process: RCP-Out

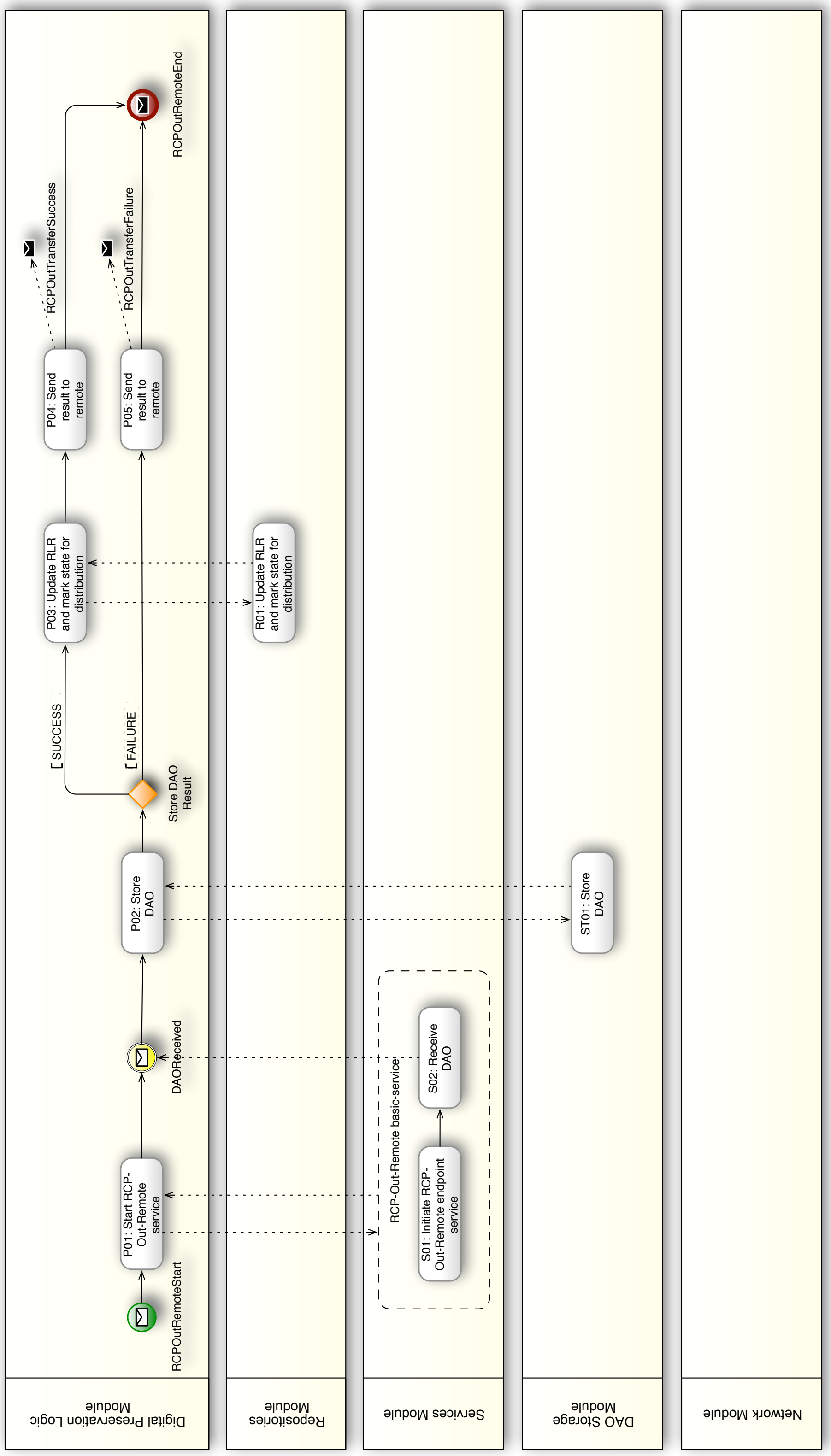
Digital Preservation Logic Module
Repositories Module
Services Module
DAO Storage Module
Network Module
RCPOutRemoteStart
P01: Start RCP-Out-Remote service
DAOReceived
P02: Store DAO
Store DAO Result
[SUCCESS
[FAILURE
P03: Update RLR and mark state for distribution
P04: Send result to remote
P05: Send result to remote
RCPOutTransferSuccess
RCPOutTransferFailure
RCPOutRemoteEnd
R01: Update RLR and mark state for distribution
RCP-Out-Remote basic-service
S01: Initiate RCP-Out-Remote endpoint service
S02: Receive DAO
ST01: Store DAO

Figure 3.18: Reliable Copy Process: RCP-Out-Remote

RCP-Out-Remote and RLR

After creating an additional replica of a DAO on a remote node, the Remote Location Repository (RLR) on the receiving node is updated by adding the replica to the list of DAOs stored on this location (step P03 in the ECP-Out-Remote process). This entry needs to be also updated on all other nodes throughout the DSN. For this, the entry will be marked as needing dissemination, and at the next running of the State Dissemination Process will be sent out to all nodes in the DSN.

Inbound Transfer

The inbound transfer direction is used when we want to retrieve DAOs or parts of DAOs stored on remote nodes, for which the RCP-In process, shown in Figure 3.19, is responsible.

The process starts when receiving the RCPInStart message, containing the DSN and the URI of the DAO as parameters, and for which we need to retrieve either some parts of, or the whole object. The transfer instructions are stored in the copy job, which the process retrieves from the CJR in the first step (P01-R01). Next, the RCP-In basic-service is called (P02-S01), retrieves the DAO parts (S01 - Remote-S01), and stores them to temporary storage (if the transfer was successful). Afterwards, the main process is informed about the outcome, which can be either successful or unsuccessful. The RCP-In process closes the job with the result of the transfer (P03-R02), and depending on the process that created the copy job, informs the DAO Repairing Process (P04) or the Node-Lost Process (P05).

3.5.10 State Dissemination Process

The *State Dissemination Process (SDP)* is a periodically running process that checks if there where changes made to the Replica Location Repository (RLR) and the Node Information Repository (NIR) that need to be published to other nodes in the DISTARNET Sub-Network (DSN). Figure 3.20 depicts the SDP.

The process is started with the SDPStart message, which contains the name of the DSN for which the process should run as the node can be part of more then one DSN. As the first step, the process retrieves any state information that needs to be disseminated from the Repositories Module (P01-R01). Next, it retrieves the members

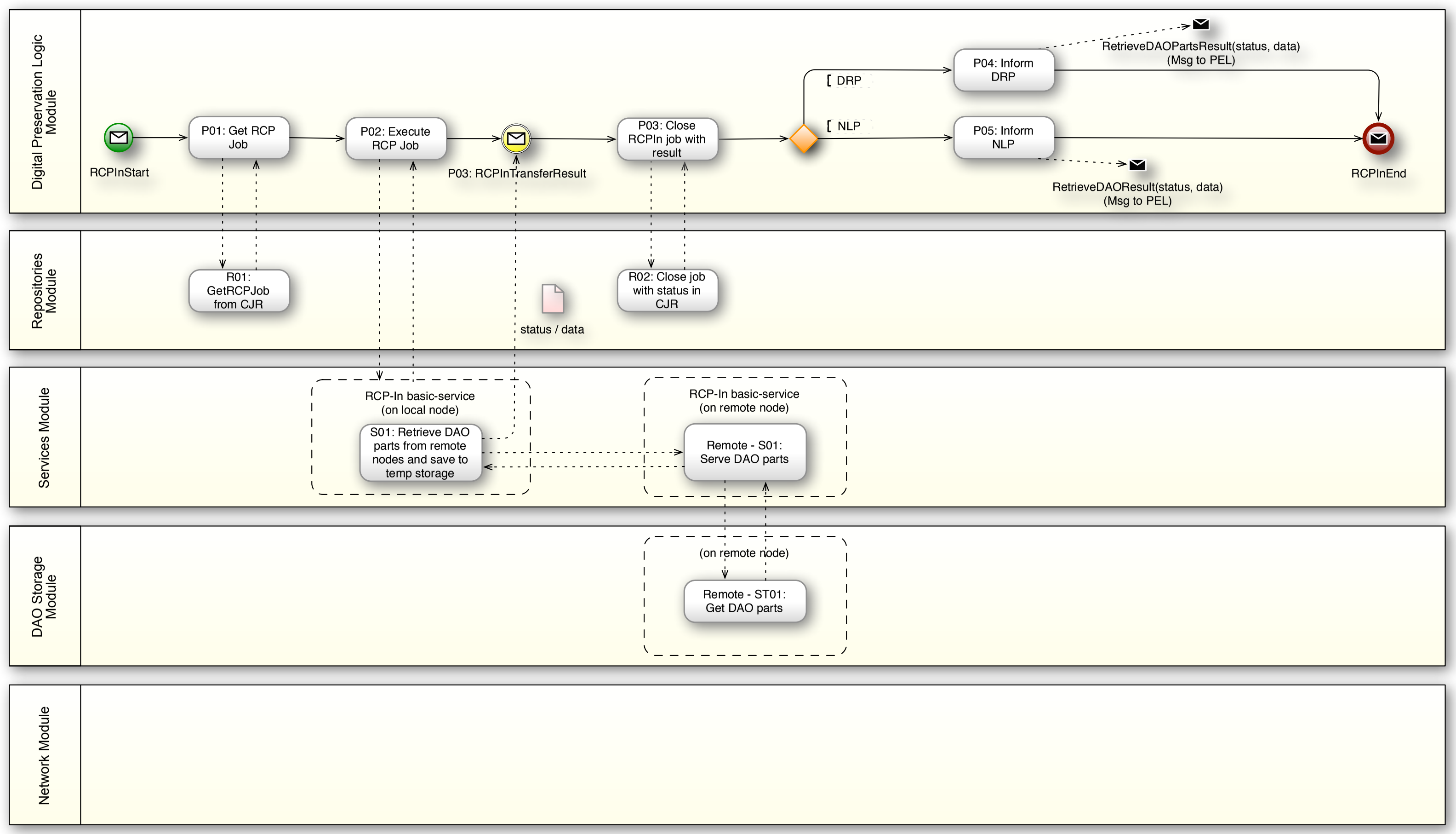
Digital Preservation Logic Module
Repositories Module
Services Module
DAO Storage Module
Network Module
RCPInStart
P01: Get RCP Job
P02: Execute RCP Job
P03: RCPInTransferResult
P03: Close RCPIn job with result
[DRP
[NLP
P04: Inform DRP
P05: Inform NLP
RetrieveDAOPartsResult(status, data) (Msg to PEL)
RetrieveDAOResult(status, data) (Msg to PEL)
RCPInEnd
R01: GetRCPJob from CJR
status / data
R02: Close job with status in CJR
RCP-In basic-service (on local node)
S01: Retrieve DAO parts from remote nodes and save to temp storage
RCP-In basic-service (on remote node)
Remote - S01: Serve DAO parts
(on remote node)
Remote - ST01: Get DAO parts

Figure 3.19: Reliable Copy Process: RCP-In

of the DSN from the NIR (P02-R02), and calls the PubSub basic-service with the list of DSN members, and states that need to be disseminated (P03). The PubSub basic-service then opens a connection to all remote nodes in the DSN and sends the states. This will be received on the remote nodes by the PubSub basic service, which will write the received states to the corresponding repositories (R03).

RLR entries: When the RLR is updated for a DAO (e.g., a replica was created on a remote node), then this entry will be marked for dissemination. The next time the SDP runs, it will send out the updated entry with the additional remote location to all nodes in this entry. After the dissemination, all nodes storing a replica of the DAO will have information regarding storage locations of all other replicas.

NIR entries: When a node updates its entry in the NIR (e.g., free storage space), it also marks this entry for dissemination. The next time when the SDP runs, it will send out this information to all other nodes in the DSN.

3.6 DISTARNET Data Model

To encompass all the described general requirements for a long-term preservation system, we need to have a flexible, and in the future extendable data model.

The concepts used in the design of the DISTARNET Data Model are partly based on the "DELOS Digital Library Reference Model" [84], which conceptually describes the structure of the *Digital Archive Resource Domain* and the *Content Domain* shown in the UML diagram in Figure 3.21. The UML digram represents only a small section of the reference model as a whole. It describes the *Resource* concept, and its specialization the *Information Object* concept, which together represent a very powerful concept for a data model as it allows to express many of the "features" we need for our data model (e.g., collection/subcollection, links, annotations, etc.).

3.6.1 DISTARNET Archival Object

In the context of DISTARNET the term *DISTARNET Archival Object (DAO)* is used to denote a container holding an *Information Object*, where the Information Object is consisting of a Data Object (e.g., image, audio/video, text document, etc.) and the corresponding representation information, or some other kind of metadata. This metadata

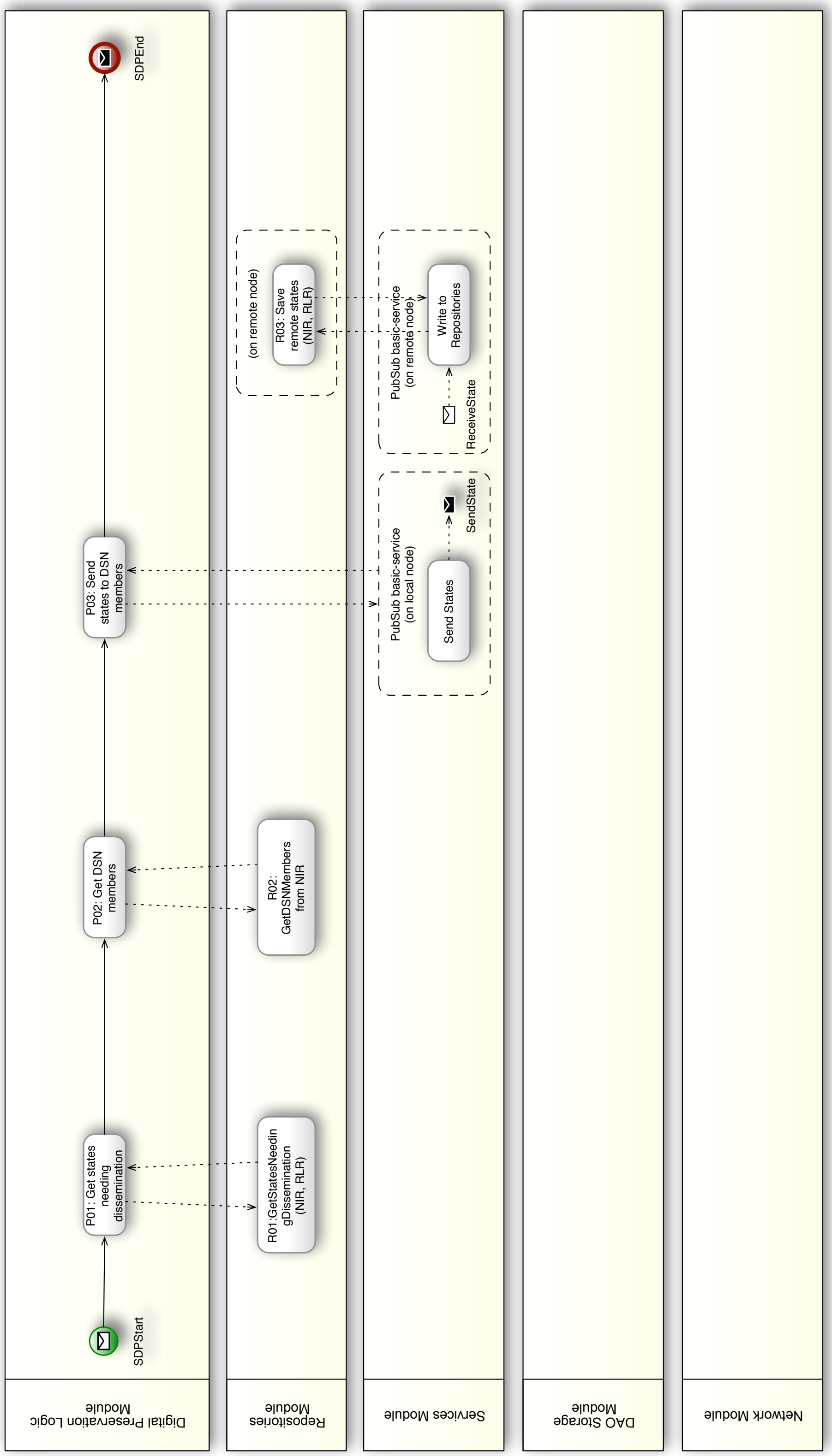
Digital Preservation Logic Module
SDPStart
P01: Get states needing dissemination
P02: Get DSN members
P03: Send states to DSN members
SDPEnd
Repositories Module
R01:GetStatesNeedingDissemination (NIR, RLR)
R02: GetDSNMembers from NIR
(on remote node)
R03: Save remote states (NIR, RLR)
Services Module
PubSub basic-service (on local node)
Send States
SendState
PubSub basic-service (on remote node)
ReceiveState
Write to Repositories
DAO Storage Module
Network Module

Figure 3.20: State Dissemination Process (SDP)

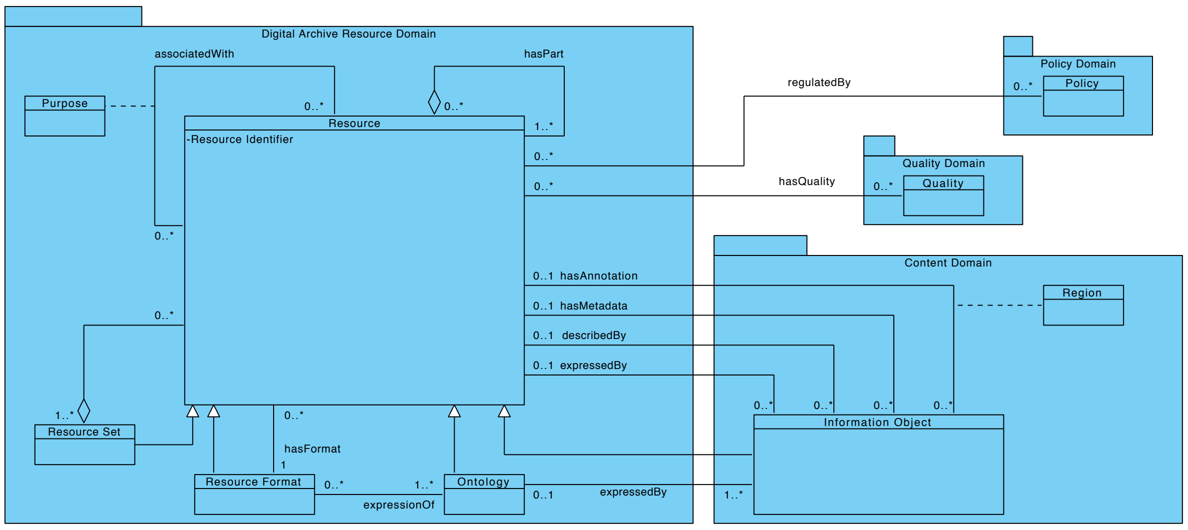

Figure 3.21: UML Diagram of the Digital Archive Resource and Content Domain

can provide additional descriptions for an information object (e.g., annotations), the descriptions of relationships between information objects (e.g., links) or information about collection/subcollection sets. Furthermore, different data supporting various aspects of the system will also be contained in the Distarnet Archival Object.

DISTARNET distinguishes between mutable and immutable content [85]: First, the digital objects that are to be archived (e.g., images, audio/video, text documents, etc.) that cannot be modified once created (read-only). Second, the metadata of the archived digital objects which may exist in several versions and which can be modified (e.g., annotations pertaining to some archived digital object; read-write semantics).

The data model used in DISTARNET allows the archiving of complex data objects, i.e., objects which are composed of or are parts of other data objects. Figure 3.22 shows the DISTARNET Logical Data Model using UML notation. Here we can see that in DISTARNET every container stores one information object characterized by its type. To represent, for example, an annotation for an archived image, we create an DISTARNET Archival Object of the corresponding type, which contains the annotation, and make a link to the DAO containing the image – note that an annotation can be anything from text to a full-fledged DISTARNET Archival Object (which again has its own metadata and annotations).

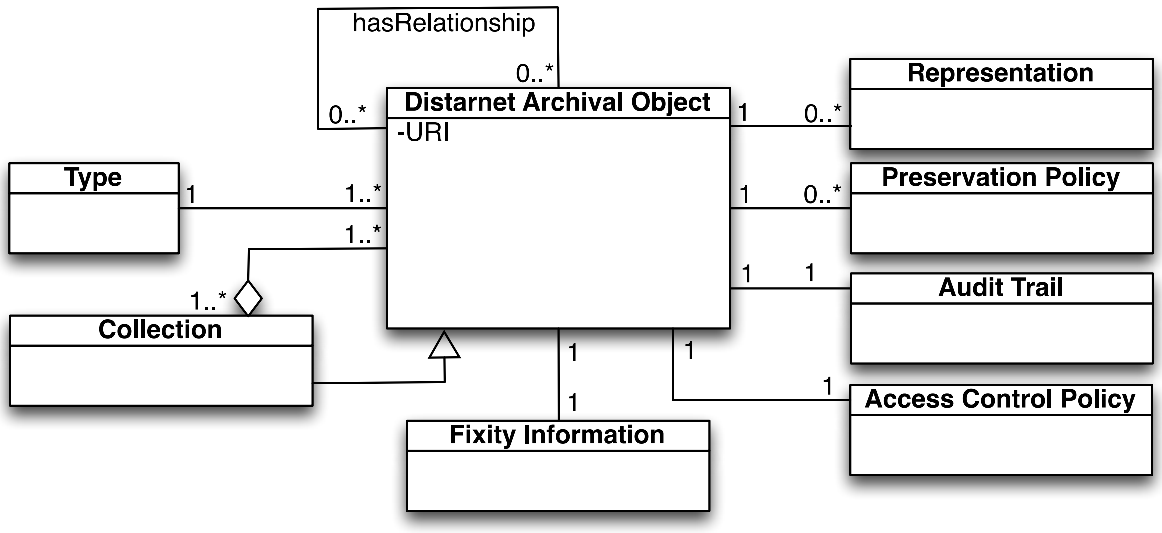

Figure 3.22: Logical Data Model for DISTARNET 2.0 in UML Notation

Representations

Every DAO has a unique identifier (URI), a type, and a set of key descriptive properties. It consists in its simplest form as an aggregation of representations. The representations provide a reference to the different representations of the digital object (e.g., thumbnail version of an image).

Relationships

Any two DAOs can be characterized by a set of relationships. These can represent, for example, in the case of a complex object, the different parts that an object is comprised of, e.g., the different pages of a book, or in the case of an annotation, the DAO that is annotated.

Collection

Every DAO can be part of a collection or a subcollection, where a collection is defined as a subclass of a DAO.

Preservation Policy

Every DAO can contain preservation policy information, which further describe any specific requirements that need to be met during the preservation, e.g., the number of replicas.

Audit Trail

Each DAO possesses an audit trail, which documents everything that has happened to the DAO during the preservation, e.g., data format migration.

Access Control Policy

Contains the access control policy for the DAO.

Fixity Information

Contains the the fixity information for the DAO, e.g., checksums of the bitstream referenced in the representation.

3.6.2 Archival Information Package (AIP)

The container storing the information object corresponds to the Archival Information Package (AIP) described in the OAIS Reference Model in Section 2.2. According to the OAIS reference model, the AIP is made of Content Information (CI), with the actual digital object that is been archived, and the Preservation Description Information (PDI) containing the corresponding metadata. Beside these, important metadata is also stored in the Representation Information (part of the CI), namely the Structure and Semantic Information of the archived digital object.

In DISTARNET 2.0 the digital object and their representation information are being treated equally. This is not contradictory since in the OAIS reference model, the representation information is just a special form of a data object (the digital object being archived). This is also why the metadata from the representation information (structural and semantic information) can be treated equally the PDI.

For example, information such as "user generated annotations", which are created over time, are not the original data objects that where archived, they are treated equally since they provide additional description information and need to be preserved as such alongside the originally archived data objects. For the future readability and interpretability of the digital object, the representation information metadata is very important. Following these requirements, a simplified model of the data-container for Distarnet 2.0 can be derived as depicted in Figure 3.23.

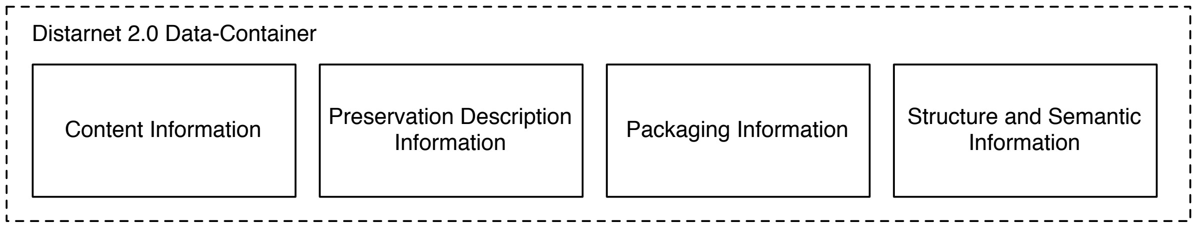
Distarnet 2.0 Data-Container
Content Information
Preservation Description Information
Packaging Information
Structure and Semantic Information

Figure 3.23: Simplified Data-Container for Distarnet 2.0

3.7 Summary

We have begun this chapter by presenting the general requirements for a long-term digital preservation system. Here, we have discussed the different overall system and user functionalities that a system should provide, and also based on the OAIS Reference Model, the different requirements on the functional entities.

Following the general requirements, we presented and discussed our proposed system model for a distributed long-term digital preservation system. The scope of the described system model corresponds mainly to the general requirements category "Data Management and Archival Storage". The system model describes a fully distributed, fault-tolerant archiving environment. The archiving environment provides autonomic behavior which is governed by preservation policies. We describe and discuss the processes that provide self-configuration, self-healing, and self-optimization, and how through their execution the system provides dynamic replication, automated consistency checking, and recovery of the archived digital objects.

Finally, we describe a highly flexible data model that we have developed that together with the specification of the sophisticated management processes, provides support for complex data objects, user generated annotations, collections, and arbitrary links.

In the next chapter, we describe and discuss the system architecture of our prototype implementation, which is based on the concepts discussed in this chapter.

4

DISTARNET System Architecture and Implementation

In Chapter 3 we have presented and discussed novel concepts for a distributed long-term preservation system for digital data, with a strong focus on long-term preservation as required by the preservation community. These concepts resulted from the combination of distributed, autonomic, and process oriented computing, with requirements from the digital preservation community regarding special user, system, and metadata functionality and needs. As a result, the described system model, consisting of a data model for complex information objects, and a number of processes was developed, providing dynamic replication, consistency checking, and automated recovery of the archived digital objects utilizing autonomic behavior governed by preservation policies without any centralized coordinator in a fully distributed network.

We have taken the described data model and processes, and used them to develop one possible implementation of these concepts. This resulted in the development of a prototype of the DISTARNET System, which system architecture we are describing in this chapter.

Before we begin with the actual description of the system architecture, we will first discuss additional requirements imposed on the implementation. These requirements where already partly discussed in the previous chapter, but as they have direct implications for the implementation, are again discussed in the new context. Following this discussion, we then describe the main architectural concepts, and libraries used in the implementation, after which we will begin with the description of the system architecture of our implementation.

4.1 Requirements for the Implementation

Beside the requirements discussed in the previous Chapter regarding modularization and fault-tolerance on infrastructure and content level, in a long-term preservation system, where the main goal is to preserve data over decades, implies that the system itself needs to be also used, maintained, and extended over the same period of time. This is an important distinction in terms of long-term use, maintainability, and extendability, compared to shorter lived systems. As such, these additional requirements for the running system need to be also taken into consideration for the design and implementation of a running system.

Long-Term Use

Many implementation choices will be impacted by this, namely the minimization of external dependencies, i.e., the balance between functionality provided through external dependencies versus a solution where everything is implemented in the system from scratch. We will use the term "external dependency" to denote any code or library that is maintained outside of this project, and thus is external to our area of influence.

Maintainability and Extendability

The requirement of long-term use implies the need for adaptability of the system to changes that can present themselves in the future, and which are unknown at the present time. As such, the implementation needs to provide as much flexibility as possible to allow the system to be easily adapted to future new need.

Node Engine Fault-Tolerance

As a distributed long-term preservation system, we need not only to focus ourselves on providing fault-tolerance in regard to the data we want to preserve, but also regarding the application providing this functionality. The former will be provided by the preservation processes, as discussed in Section 3.3. The later will be discussed in this section, and needs to be taken into consideration for the implementation.

In the following, we take a bottom-up approach, where we first discuss Node Engine fault-tolerance strategies that need to be taken into account for the implementation. Afterwards, we take a look at the implementation possibilities that provide long-term maintainability and extendability of the system. Thereafter we look over some considerations regarding the minimization of external dependencies. Last but not least we round up the discussion with a summary, where we discuss the strategy for the implementation of the system that we follow thereafter.

4.1.1 Node Engine Fault-Tolerance

Building on Section 2.5, we will resume the discussion we have started in Section 3.3, and discuss the third class of faults, the *Node Engine* class. Table 4.1 shows again the three DISTARNET fault classes that we have defined, their effects when they occur, how they will be detected, and what the recovery actions of the system will be. Previously we have already covered the *Distributed Infrastructure Class* of faults where we have discussed how the system will deal with remote node failures, and what processes will be used to recover. Also, we have discussed the *Content Class* of faults, where we have seen how the system will recover from faults regarding the content, e.g. corrupt archived digital data, and also what processes will be involved. In the following, we will discuss the *Node-Engine Class* of fault, and look what strategies can be followed to provide Fault-Tolerance on the *Node-Engine* level, i.e., to allow a fault-tolerant execution of the processes which are used in the previous two classes. These strategies will need to be inherently provided by the implementation. This requirement will have a strong influence on any further discussion and decisions regarding the implementation design.

The Node-Engine class of faults includes faults that can occur during the execution of the DISTARNET processes, and their interaction with the modules a node is composed of. We will concentrate us on the *Crash*, *Omission*, and *Timing* faults which we introduced in Section 2.5, which can occur during the execution of the process requests.

To help us in the discussion, we will further divide the requests initiated by the DISTARNET processes by the scope, into a *Network Scope*, and a *Local Scope*. The Network Scope includes processes that involve interaction with other nodes in the network, and the Local Scope where the processes are confined to the local node. We will only look one-sidedly on these requests, from the point-of-view of the local node. The same behavior holds true for all nodes in the network as they are built all

Fault Class	Failure	Detection / Reaction
Distributed Infrastructure	**Hardware Problems** Failure (power, hardware, etc.) Disaster (natural, fire, etc.) => *Node Loss*	***Modules Involved:*** DP Logic ***Detection:*** PNCP ***Reaction:*** Node Lost Event; Repository updt, ADRP
	Network Problems intermittent/periodic connection loss, etc. => *Remote Node Dependability*	***Modules Involved:*** DP Logic ***Detection:*** PNCP ***Reaction:*** Repository updates, ADRP
Content	Localized hardware problems, malicious acts, etc. => *DAO Corruption*	***Modules Involved:*** DP Logic, DAO Storage ***Detection:*** PICP ***Reaction:*** Repository updates; DAO update; ADRP
	Format obsolescence => *DAO Representation Unreadable*	***Modules Involved:*** DP Logic, DAO Storage ***Detection:*** PICP ***Reaction:*** DFMP
Node Engine	A problem occurring during the execution of a DISTARNET process => *Process Execution Failure*	***Modules Involved:*** DP Logic ***Detection:*** process execution logic ***Reaction:*** execution of corresponding recovery process

Table 4.1: DISTARNET Fault Classes, their Effect, Detection, and Recovery Actions

the same.

Local Scope

Every process will be composed of different building blocks, which when executed fall into one of the following two: (1) side-effect free building blocks, and (2) building blocks with side-effects.

The (1) *side-effect free building blocks* are either stateless, or only the local state of the building block is changed. A stateless building block, for example, is a piece of code that calculates a checksum. We would execute the building block by providing an input value, and as a result, we would get the checksum. In case of failure of such an building block, we can just simply retry the execution. In the second case were some internal state is changed, for example if the building block is called subsequently and needs to store some values temporarily, we could employ the checkpointing strategy, and save the local state periodically, and then in case of fault, restore the state, and allow to retry the execution from the last saved state.

The (2) *building blocks with side-effects*, have side-effects in the form of global state changes, for example, if the building block is an abstraction of a database, and the state changes are the writes we execute against the database. Here, we can also employ the checkpointing strategy, to save and restore the state in case of a fault, but this will need to be implemented on a global level as other processes running can have dependencies on the global state that we would change in case of a restore. As we have a modular system design, global state means in this case global at the module level.

Network Scope

Everything we have said so far, will also hold for processes running in a network scope, as they also have parts that run locally. So additionally, we have interactions with other nodes, during which the crash, omission, or timing faults of a remote node, can have adverse effects to the running process, and cause it to fail. The strategy here will be to design every call to a remote node in an asynchronous way, and add a time-out condition. Triggered by the time-out condition, we must provide an alternative path of process execution which will deal with the problem, either by simply retrying, or by further escalating the problem.

Beside the *Retry* and *Checkpointing* strategies discussed so far, we can additionally employ the *Software Rejuvenation Strategy* on the module level, where we would restart each module, and clean out the internal states.

Fail-Stop

In the case of a fault from the Node Engine class, where in spite of any recovery attempts the node engine is not able to resume proper functioning, the system will follow the fail-stop strategy. In this case, the system will send out notifications to inform that it is not able to recover from a failure and stop. In this instance, external assistance will be needed. Any unsuccessful recovery from a failure in the *Distributed Infrastructure* or *Content* class will only produce notifications, without stoping the system.

Multiple Faults

As a basic requirement, for the system to be able to execute any processes, the Node Engine needs to function properly. In the case of multiple faults where the combination also includes the node engine, first the node engine itself needs to recover before any other recovery processes can be executed. Any combination of *Distributed Infrastructure* and *Content* faults can be recovered from at the same time, as long as there are a minimum number of remote nodes available that can be used as alternatives for the storage of the archived content, and there is at least one healthy replica available in the network from which the content can be recovered from.

4.1.2 Maintainability and Extendability

Implied by the long-term use of the system, the maintainability and extendability requirement, needs to be met through an implementation design, supporting adaptability to changes that can present themselves in the future. This can be achieved by employing a *process-driven application design*, where on one side, we have well defined *processes* that can be exposed and reused, and on the other side *atomic services* that will be consumed during the execution of these processes.

Additionally, this service orientation will be accompanied by *modularization*, which means that the different services will be organized, implemented, and provided by well defined and independent modules. This is motivated by the need to allow parts of the system to be exchanged for newer implementations in the future, to allow to be adapted to changes in the software ecosystem composing and/or surrounding the preservation system. Furthermore, this modularization is also a requirement that comes down from the Application-Level Fault-Tolerance Strategy discussion.

Embedded Process Engine vs. Custom Process Execution Logic

In the following, we will discuss the different alternatives for the implementation of a process-driven application, where beside a strong separation of application flow from application execution, we have additionally strong fault-tolerance requirements that need to be taken into account as discussed earlier. For the implementation, we will need a system providing us with the ability to define the process flow, execution, control, and compensation. For this, we have three general possibilities to our disposal: (I) build the whole system on top of an integrated system providing process execution, (II) embedding an existing process engine into the system, and (III) implementing a custom process execution logic. Before we can make such a decision, we need to take a look at the advantages and disadvantages of all the possibilities.

In the first case, we would build the system on top of an integrated system providing process execution such as IBM WebSphere[1]. This would allow us to define our process flows, and then deploy them to the process engine, which will be responsible for their execution. WebSphere would provide a full ACID transaction support for processes, both for short-running (one transaction end to end) and long-running processes (multiple transactions). Transaction boundaries can be modified to allow the grouping of multiple steps in a process into one transaction. Additionally, it supports flexible compensation of business processes as defined in the WS-BPEL specification. The main drawback would be to create a large external dependency and the imposed monolithic design.

In the second case, where we would embed an existing process engine, we would have the advantage of integrating it with our overall modular system design (unlike the monolithic design which would be imposed by the first case), with the added advantages provided by a process engine. These include the advantage to allow us to define our process flows, and then deploy them to the process engine, which will be responsible for their execution, and persistence of the process instance states during their execution. In the case of jBPM[2], Activiti[3], and Bonitasoft[4], we can define the processes in the BPMN 2.0[5] notation which can then be deployed. They generally provide a smaller feature set as compared to WebSphere, especially the ACID transaction support for processes. Also, one of the drawbacks is that an embedded process engine will need an additional custom software layer to be able to talk to the rest of the system.

In the third case where we implement a custom Process Execution Logic, the implementation will not be able to support process definitions in the form of the BPMN 2.0 notation, because such a feature would be coupled with a high implementation cost, which would on the other side be somewhat alleviated by the fact, that the custom implementation would be able to directly communicate with the rest of the system. This means that the process flows will need to be "hard-coded" in the programming language used in the implementation. One can now argue that by doing so, we lose the very flexibility that we wanted to have in the first place, by using a process engine. In the case of service-oriented programming-in-the-large where there are possibly dif-

[1]`http://www.ibm.com/websphere`
[2]`http://www.jboss.org/jbpm`
[3]`http://activiti.org/`
[4]`http://www.bonitasoft.com/`
[5]`http://www.bpmn.org/`

ferent suppliers, or different implementations of the same service, this holds true. But in the case of our system, where we apply the programming-in-the-small paradigm, and where every service that we want to call, needs to be first implemented in the system, this does not hold true. Any change in the process flow description, where old service calls are changed, or new service calls are added, need also to be reflected in the implementation of these services in the system. There are certainly some services that theoretical can be implemented outside of the system by an external provider, and consumed in the SOA manner, e.g., calculating the checksums, but the added cost of sending the large digital files over the network, or the implications of sending content to an external provider, renders this non-feasible.

Another advantage of embedding an existing process engine lies in the persistence of the process instance states during their execution. This allows us to restore all process instances after a crash, and resume their execution.

This is a feature that will need to be also provided in the case where we would implement a custom process execution logic. The main challenge in either case lies in the implementation design of the services consumed by the processes so that they will allow to be resumed after a complete system crash.

We will come back to this question in the summary of this section.

4.1.3 Long-Term Use

In the context of long-term use of a system, an important factor that needs to be taken into account are the external dependencies. They need to be minimized as to be able to allow and provide a long-term deployability, maintainability, and extendability of the system. The term "external dependency" denotes any code or library that is maintained outside of this project, and thus is external to our area of influence. Although for DISTARNET, we use only open-sourced external dependencies, an active maintenance is important to guaranty future compatibility. If any external dependency becomes unmaintained, then this dependency is threatening the long-term working of the system, and needs to be exchanged. Table 4.2 list the external dependencies per module and Table 4.3 provides a summary of the overall external dependencies.

Module	External Dependency	Usage Description
User Interaction Module	Scala / Java / JVM	Programming and runtime environment
	Akka	Communication Framework
	Netty / Unfiltered	REST API (Netty is also used by Akka)
Digital Preservation Logic Module	Scala / Java / JVM	Programming and runtime environment
	Akka	Communication Framework
	jBPM or Activiti or Akka-FSM	Process Engine
	MongoDB	Repositories / Process Engine
Services Module	Scala / Java / JVM	Programming and runtime environment
	Akka	Communication Framework
Network Module	Scala / Java / JVM	Programming and runtime environment
	Akka	Communication Framework
DAO Storage Module	Scala / Java / JVM	Programming and runtime environment
	Akka	Communication Framework
	Jena-Core	API for accessing the triple store
	Jena-TDB	Triple Store

Table 4.2: External Dependencies per Module

External Dependencies
Scala / Java / JVM
Akka
Jena-Core
Jena-TDB
jBPM or Activiti or Akka-FSM (provided in Akka)
MongoDB
Netty (used also by Akka) / Unfiltered

Table 4.3: External Dependencies Summary

Secondary External Dependencies

The Figures 4.1, 4.2, 4.3, and 4.4 show the secondary external dependencies of the different embeddable process engines. If we take into account that all three engines have a comparable feature set, we can see that in the case of Bonitasoft, when compared to Activiti and jBPM, we have an additional large set of secondary dependencies, which make this process engine unsuitable for our needs.

4.1.4 Summary

In the context of a long-term deployment of the system, we need to weight the advantages provided by feature rich external dependencies against the disadvantages of increasing our external dependencies. External dependencies can negatively impact the long-term deployability and maintainability of the system if they become unmaintained, in which case we would need to exchange them. Depending on the usage, i.e., on the location where in the system it is used, this exchange can be more or less complex. If we would use a monolithic design for the system, then the complexity of such an exchange would rise additionally. Through the modularization of the system, our goal is also to make this kind of exchange less complex. Despite the modularization, we will still have some core functionality of the system, that although possible to be exchanged if needed, plays a large central role. For such core functionalities we would tend to use our own implementation if it would be feasible implementation cost wise.

Summing it up, a feature rich platform like WebSphere is not an option, in spite the feature richness of the platform as this would demand a very large external dependency, and also confine us to a monolithic design imposed by the underlying system.

Generally speaking, implementing everything by ourselves is also not an option as the cost of doing so would be prohibitively high. As such we will create as little external dependencies as possible while at the same time try to reuse them as much as possible. Additionally we will be employing a modular design that will allow the exchanging of certain parts of the system in the future, in case the external dependencies are not available anymore.

In the case of embeddable processes engine versus custom Process Execution Logic, we come to the conclusion that if the cost of developing a custom Process Execution Logic, with the features described earlier is feasible, then we would tend towards a

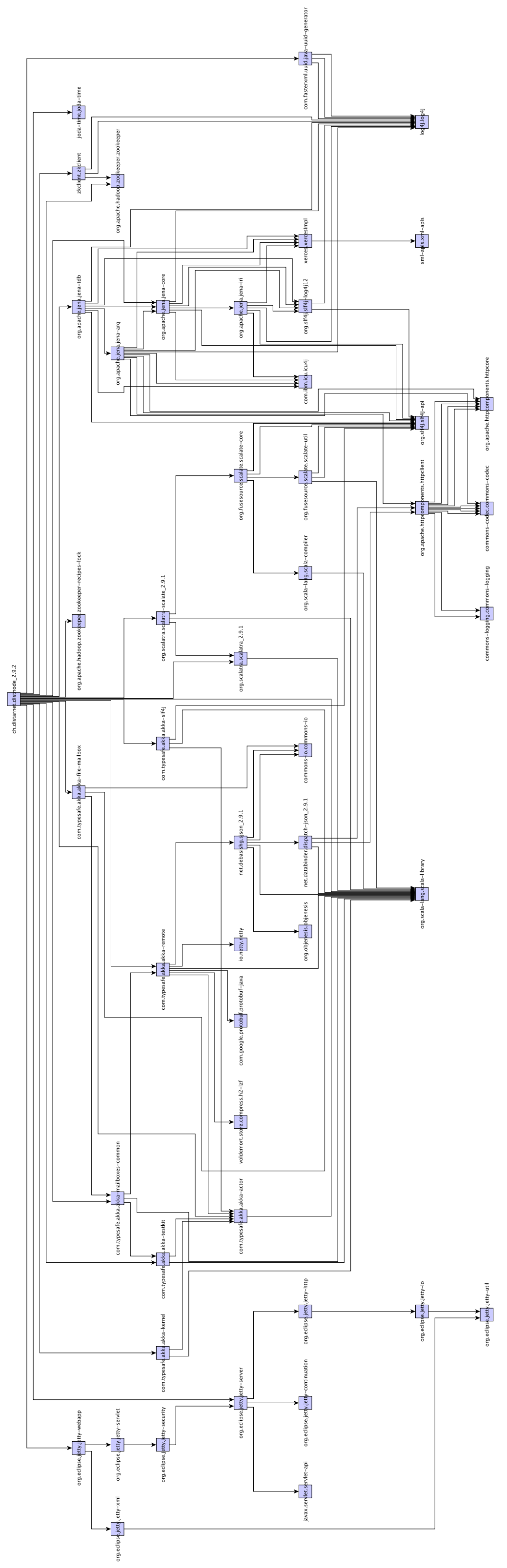
com.fasterxml.uuid.java-uuid-generator
log4j.log4j
joda-time.joda-time
zkclient.zkclient
org.apache.hadoop.zookeeper.zookeeper
xerces.xercesImpl
xml-apis.xml-apis
org.apache.jena.jena-tdb
org.apache.jena.jena-core
org.apache.jena.jena-iri
org.slf4j.slf4j-log4j12
org.apache.jena.jena-arq
com.ibm.icu.icu4j
org.slf4j.slf4j-api
org.apache.httpcomponents.httpcore
org.fusesource.scalate.scalate-core
org.fusesource.scalate.scalate-util
org.apache.httpcomponents.httpclient
commons-codec.commons-codec
org.scala-lang.scala-compiler
commons-logging.commons-logging
org.apache.hadoop.zookeeper.zookeeper-recipes-lock
org.scalatra.scalatra-scalate_2.9.1
org.scalatra.scalatra_2.9.1
ch.distarnet.distnode_2.9.2
com.typesafe.akka.akka-slf4j
commons-io.commons-io
com.typesafe.akka.akka-file-mailbox
net.debasishg.sjson_2.9.1
net.databinder.dispatch-json_2.9.1
org.scala-lang.scala-library
org.objenesis.objenesis
io.netty.netty
com.typesafe.akka.akka-remote
com.google.protobuf.protobuf-java
voldemort.store.compress.h2-lzf
com.typesafe.akka.akka-mailboxes-common
com.typesafe.akka.akka-actor
com.typesafe.akka.akka-testkit
org.eclipse.jetty.jetty-http
org.eclipse.jetty.jetty-io
org.eclipse.jetty.jetty-util
com.typesafe.akka.akka-kernel
org.eclipse.jetty.jetty-server
org.eclipse.jetty.jetty-continuation
org.eclipse.jetty.jetty-webapp
org.eclipse.jetty.jetty-servlet
org.eclipse.jetty.jetty-security
javax.servlet.servlet-api
org.eclipse.jetty.jetty-xml

Figure 4.1: Secondary Dependencies without any Process Engine

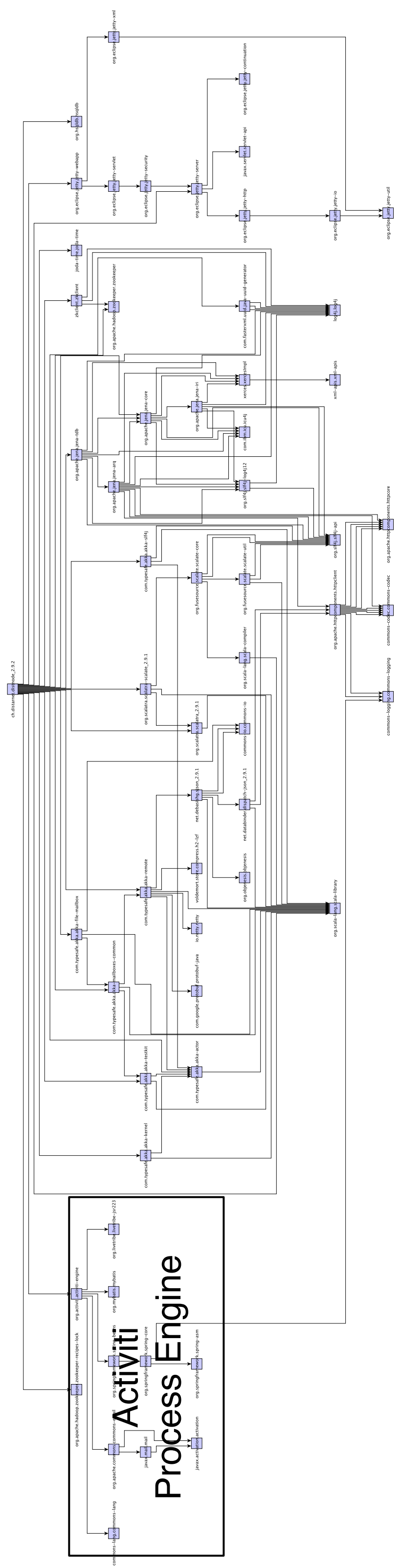

Figure 4.2: Secondary Dependencies including the Activiti Process Engine

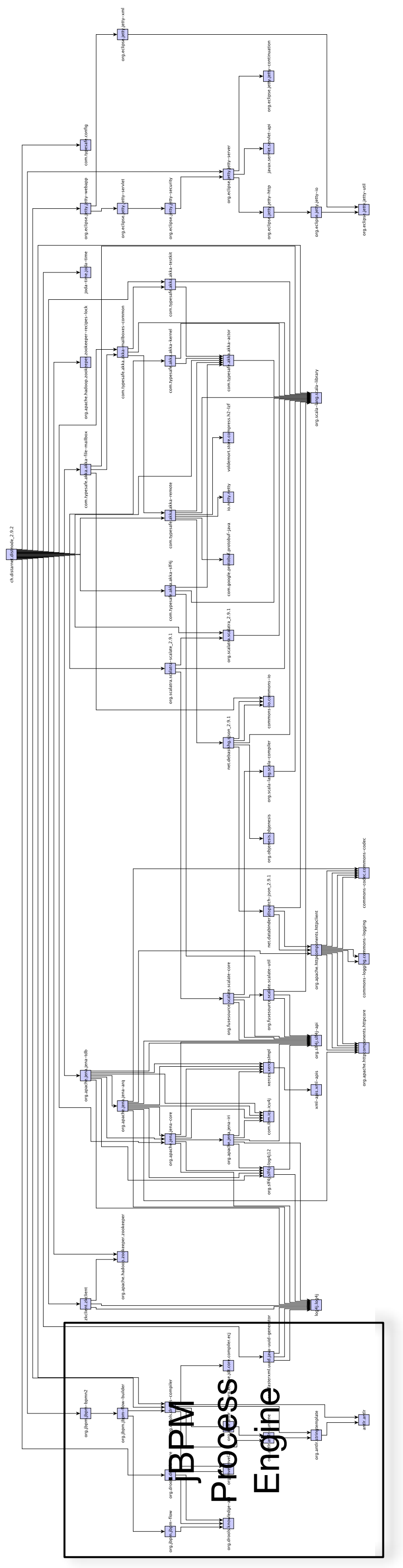

Figure 4.3: Secondary Dependencies including the jBPM Process Engine

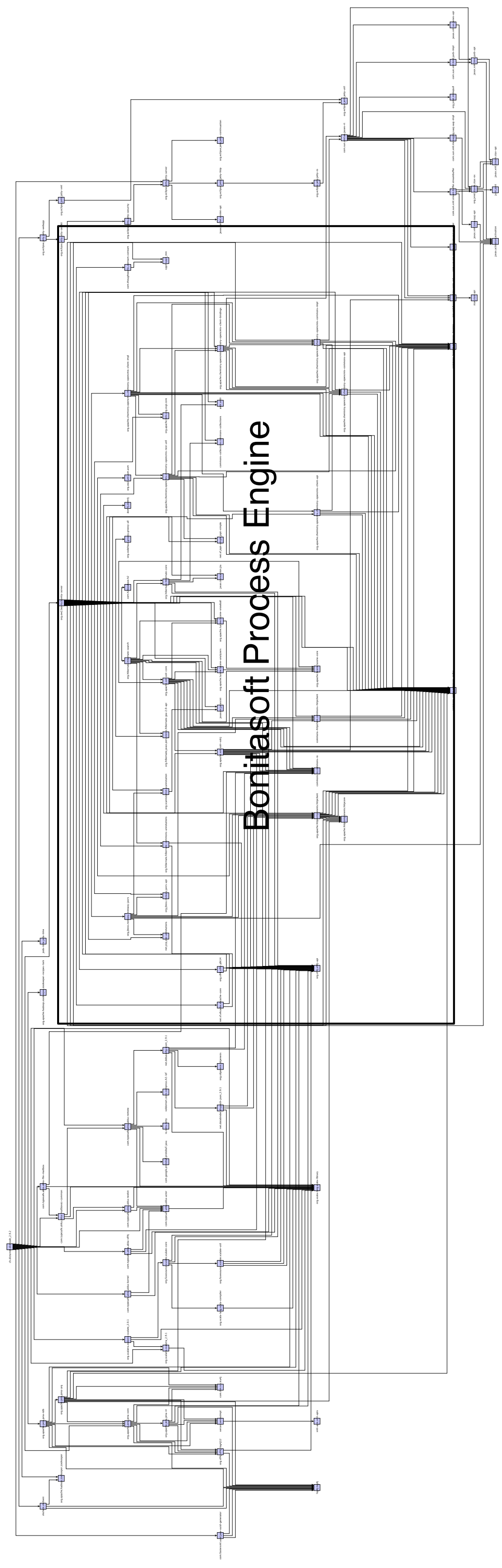

Figure 4.4: Secondary Dependencies including the Bonitasoft Process Engine

solution where we would implement our own custom Process Execution Logic.

4.2 Implementation-Specific Concepts, Frameworks, and Libraries

In the following, we describe the main architectural concepts, frameworks, and libraries that are used for the implementation of the DISTARNET system prototype.

4.2.1 Actor Model

The implementation of DISTARNET is heavily based on the *Actor model* concept, and as we will see in later in this chapter, used at all levels.

An *Actor* is basically a model for concurrent computation execution where the *actor objects* encapsulate state, behavior, and have their own thread of control. The communication with other objects is done asynchronously by passing immutable messages. For this, every actor possesses its own mailbox for storing messages, implemented as queues. The concept of actors goes back to the work of Hewitt et al. [86].

Basically, every actor follows the following three simple rules:

1. The processing of messages from an actors mailbox is only done *sequentially.*
2. The *order* by which an actor sends messages out to another actor is the same order by which these messages will be processed by the receiving actor.
3. All messages that are sent out an actor must be *immutable* objects.

By deploying only one actor, we will not get a system that provides concurrent execution. To achieve concurrency, we need to deploy many actors interacting with each other.

We decided to use the Actor model for the implementation of DISTARNET because of its attributes regarding asynchronicity and concurrency. Both attributes, we believe, will gain more and more importance as the trend towards increasing CPU count and cores per CPU progresses. With the use of the Actor model, the goal is to provide a good platform, that will be able to scale well not only vertically (number of CPUs) when deployed on a node running on more performant hardware, but also horizontally (number of nodes), when used in a distributed setting.

4.2.2 JVM, Scala, and Akka

The decision towards the JVM platform as such was influenced by the long-term use requirement discussed in Section 4.1.3 as only the JVM needs to be ported on new hardware, while the rest of the application should work as is. Additionally, the JVM provides platform independence, allowing to deploy the solution on a large number of operating systems. Further, the abundance of software libraries available in the Java programming language and the widespread use were also important factors for the decision towards the JVM platform.

A number of factors contributed to the selection of Scala [87] as the programming language for DISTARNET's implementation. Scala programs compile directly to Java bytecode, and allow the calling of Java classes directly, without any intermediate interfaces or proxies. The trend towards increasing CPU count and cores per CPU rather than increasing clock speeds means that applications of all types increasingly have to rely on concurrency for performance. Functional languages are perhaps the most promising alternative for writing concurrent applications as they avoid the difficulties inherent in managing concurrent access to mutable state. Scala's support for functional programming, and paired with its seamless interoperability with Java, provided a natural fit.

As a natural progression regarding the capabilities for implementing concurrent systems in Scala is the Akka[6] framework a library written in Scala. Akka provides besides scaling up through concurrency (e.g., local actors), also scaling out through remoting (e.g., remote actors), and also additionally inherent strategies for fault-tolerance (e.g., actor supervision). The Akka framework thus allows the creation of event-driven, scalable, and fault-tolerant systems by utilizing the actor model.

The core of Akka, namely the *akka-actor*, is a very small library which is easily dropped into an existing project where asynchronicity and lockless concurrency are needed. Beginning with Scala version 2.9.10, *akka-actor* is part of the Scala language where it will replace the current actor implementation, and will not be needed to be added as a separate library.

[6] `http://akka.io`

4.2.3 Akka's Actor System

Akka's actor implementation is very lightweight, where different actors can share the same thread. All messaging, local or remote is done by sending messages to actor paths which point to the receiving actor implementation.

Actor Paths: Each actor path has an address component, describing the protocol and location by which the corresponding actor is reachable, followed by the names of the actors in the hierarchy from the root up, e.g.:

`akka://mySystem@server.com:5678/user/service/myService`.

Here, `akka` is the default remote protocol used, but others can be plugged in through a custom implementation.

As there can be any number of actor systems running inside a JVM, the name of the receiving actor system must be given, which is in the example `mySystem`.

The interpretation of the host & port part (i.e., `server.com:5678` in the example) depends on the transport mechanism used, but it must abide by the URI structural rules. The part after the port number is the actors name (i.e., `myService` in this example) embedded in the hierarchy.

Children: Each actor can create children for delegating subtasks. By doing so, it will automatically become the supervisor of these actors.

Supervisor Strategy: As a supervisor, the supervising actor delegates tasks to subordinates and therefore needs to also respond to their failures. In the case of a detected failure in a subordinate, he will send a message to his supervisor, and signal the failure. The supervisor can then decide the best course of action, ranging from resuming, restarting, and terminating the subordinate actor, and also even the possibility of escalating the failure higher up the hierarchy.

4.2.4 Netty and Unfiltered

Unfiltered [7] is a toolkit that allows to serve HTTP requests in Scala. The toolkit is server backend agnostic, meaning that it provides a consistent vocabulary for handling HTTP requests, without being tied to a certain server. The server used as the

[7] `http://unfiltered.databinder.net`

backend in our implementation is a server based on the Netty Framework [8]. Netty is a highly flexible asynchronous event-driven network application framework, which can be used to rapidly develop protocol servers. In our case, we use it to create a small http server, functioning as the backend of the Unfiltered toolkit.

The Netty Library is already used by the Akka Framework so that only the Unfiltered library is additionally needed. As Unfiltered is a Scala library, it integrates well and easy with Akka, allowing a thin implementation.

It would certainly be possible to use any other libraries instead of Netty and Unfiltered, but the goal was to reuse the available libraries as much as possible, and to create a simple implementation as possible.

4.2.5 Jena Core and Jena TDB

As we will see in Section 4.5 where we describe the implementation of the DISTARNET data model, we use RDF to represent the archived Information Objects. For working with RDF data, we use libraries from the Apache Jena[9] project. The libraries consist of the Jena Core library which provides a rich API for working with RDF data, and the Jena TDB which is a native implementation of a triple-store done in java.

There are certainly any number of other libraries which could have been used for the implementation, that provide a similar feature set. We have decided to use this particular implementation basically because of two things. First, we implemented the metadata benchmark using this library (see Section 5.1), and the resulted performance was more than satisfactory. Second, for pragmatic reasons as we had good prior experience with the library.

4.2.6 Mongo DB and Casbah

For the implementation of the different repositories we are using Mongo DB[10], a document oriented database, and the Casbah library[11] which is the official Scala API for Mongo DB. The reasoning beside the usage of a document oriented database is twofold. First, the structure of the different repositories used throughout the DIS-

[8]`http://netty.io`
[9]`http://jena.apache.org`
[10]`http://www.mongodb.org`
[11]`http://api.mongodb.org/scala/casbah/current/`

TARNET system can be represented in single tables, without any relations to other tables. Second, a document oriented database allows us to have a very flexible data model, which allows us to mirror it closely to the internally used data structures, that we can store with little effort.

4.3 System Architecture Overview

In the following, we give a short overview of the system architecture of the DISTARNET system implementation.

As discussed in Section 3.4, the proposed system model was developed as being strongly modularized, where each module provides a certain subset of the functionality needed by the whole system. Figure 4.5 shows the familiar modular design described earlier, which we have taken over for the implementation of the prototype. The figure additionally includes annotations regarding the technologies used for the implementation of the different parts.

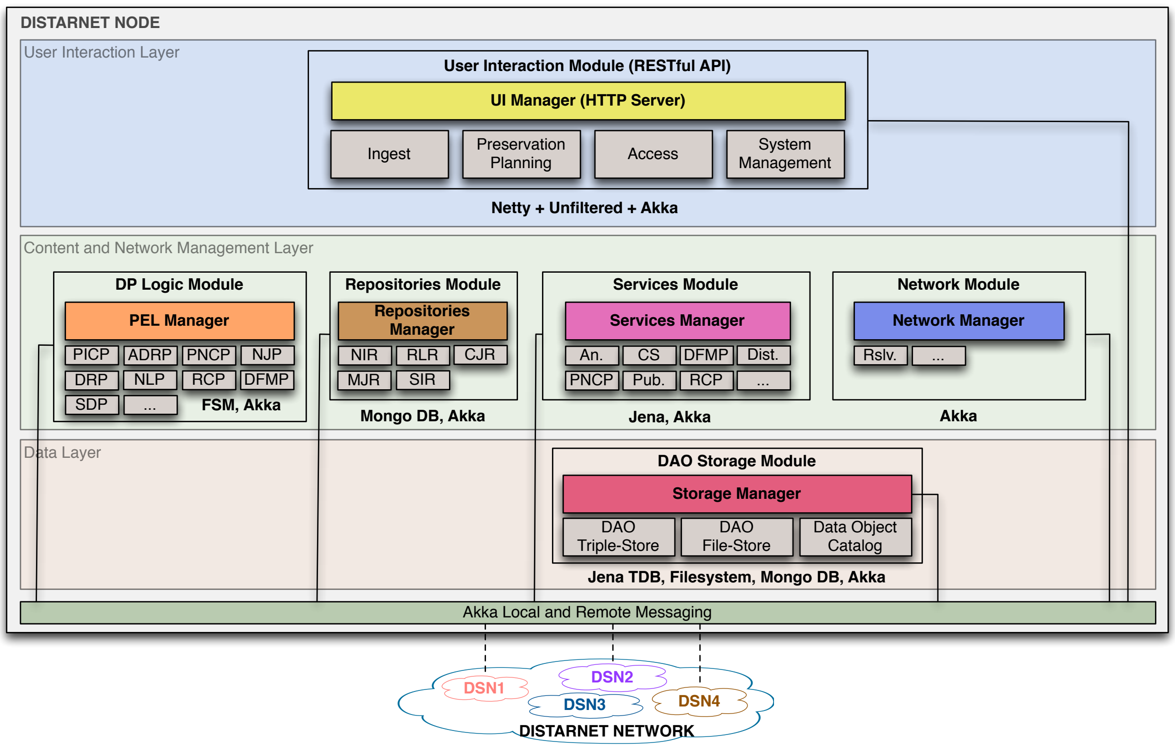

Figure 4.5: DISTARNET System Architecture Overview

Node Modules

As mentioned earlier, for the implementation of the system we are using the Scala programming language, where the modules and the parts they are made of, are implemented as *Actors* with the use of the Akka framework. The communication between *Actors* is message based, and as such also any interaction between the different modules is done by exchanging messages. The use of a message based architecture allows our implementation to provide the features described in Section 3.4 regarding modularity. Also, it facilitates the implementation in regard to the extended set of requirements that where discussed in Section 4.1, relating to long-term use, and node engine fault-tolerance. Both will be discussed in more detail when we take a deeper look at the implementation of the modules in Section 4.4.

User Interaction Module Beginning at the top we have the *User Interaction Module*. The foundation for the implementation of these modules will be implemented as a RESTful API using Netty as the HTTP server in combination with the Unfiltered library. On top of the API, the implementation of the individual user and administrative access parts (the grayed out parts) to the system can be build, which are out of the scope of the implementation.

Digital Preservation Logic Module The *Digital Preservation Logic Module* houses on one side the Process Execution Logic (PEL), and on the other side the different repositories as described in Section 3.4.2. The Process Execution Logic is a custom implementation of an execution environment using Akka, in which the DISTARNET processes are implemented as *Finite State Machines (FSM)*.

Repositories Module The Repositories Module contains besides the *Replica Location Repository (RLR)*, the *Node Information Repository (NIR)*, the *Copy Job Repository (CJR)*, and the *Migration Job Repository (MJR)* also the *System Information Repository (SIR)*. The RLR, NIR, CJR, and MJR will provide the features described in Section 3.4.2, whereas the SIR is used by the Process Execution Logic for the persistence of the states during the execution of the DISTARNET processes. All the aforementioned repositories will be implemented as actors using the Casbah API to communicate with an instance of Mongo DB used for persistent storage.

Services Module The *Services Module* includes the implementations of the basic-services which are consumed by the *Digital Preservation Logic Module*. The basic services are as before implemented as actors using Scala and Akka, and additionally those services accessing RDF data are using the Jena library.

Network Module The *Network Module* contains the network services written in Scala with the usage of the Akka framework for routing messages from the local node to remote nodes, and received messages from remote nodes to the corresponding local modules.

DAO Storage Module The *DAO Storage Module* contains the *DAO Triple-Store* which is implemented as a Jena TDB triple-store. Further, the *DAO File-Store* and the *Data Objects Catalog* are implemented as flat files on the filesystem. All three submodules use the Akka library and are implemented as actors.

DISTARNET Network

An implementation of DISTARNET will need to accommodate a broad spectrum of users as introduced in Section 1.1, e.g., institutional, corporate, scientific, etc. As such we deemed it necessary to provide one flexible implementation as supposed to building a separate implementation for each scenario. Thus, the implementation needs to be usable in a smaller and in a larger setting.

The size of a DISTARNET Sub-Network (DSN) depends on the deployment scenario. As a consequence, the size of the DSN can vary from just a few nodes to a couple of hundred or thousand nodes. This wide range of deployment scenarios will have an impact on the implementation design. For example, State Dissemination inside a DSN can be implemented with a simple publish-subscribe pattern in combination with flooding, or a more complex gossiping algorithm and a distributed hash table (DHT). The former will suffice when only smaller DSNs are assumed, and the later will be needed for a good performance when a DSN consist of thousands of nodes.

Although very large DSNs can be a possibility, the current implementation will limit itself to use cases where the size of the DSN is in the order of tens of nodes. This does not mean that the system can not be adapted to be usable with large DSNs. Through the modular design, it will be possible to extend the appropriate modules, and add additional functionality that will provide efficient support for large DSNs.

4.4 DISTARNET Modules Implementation

The implementation of the DISTARNET system at large, and modules in particular is based on using the Akka Framework, an implementation of the Actor Model, to create an event-based distributed system architecture.

At a micro level, every node is broken down into modules, where any interaction between the modules is message-based. Every module has a dedicated *Actor* functioning as the entry point for intermodule communication. Also, all the different functionalities encapsulated in the modules themselves are also implemented as *Actors*.

At a macro level, these modular nodes create together a distributed network of cooperating nodes, which again use the same kind of message-based communication.

4.4.1 Node Actor System

The modular design provided through the usage of the Actor Model with the Akka framework brings certain advantages to our implementation. Firstly, in regards to the long-term use discussion (see Section 4.1.3), it will provide flexibility and independents regarding the implementation as it allows exchanging and/or extending of each modules functionality with different implementations. Secondly, regarding the discussion of node engine fault-tolerance (see Section 4.1.1), such a modular design paired together with a message based communication between the modules, allows full decoupling of the modules, and with the supervisor strategy, provides additional finer grained fault-tolerance.

The DISTARNET implementation uses the `akka` protocol by default. The name of the Actor System is defined in the implementation of a node as `DistNodeSystem` so that the generic endpoint path takes on the form of:

```
akka://DistNodeSystem@host:port/user/endpoint.
```

Using the Akka framework, each module is represented by an actor, creating the *DISTARNET Node Actor System* as depicted in Figure 4.6. As already mentioned, all communication between the module actors is done via messaging, decoupling them

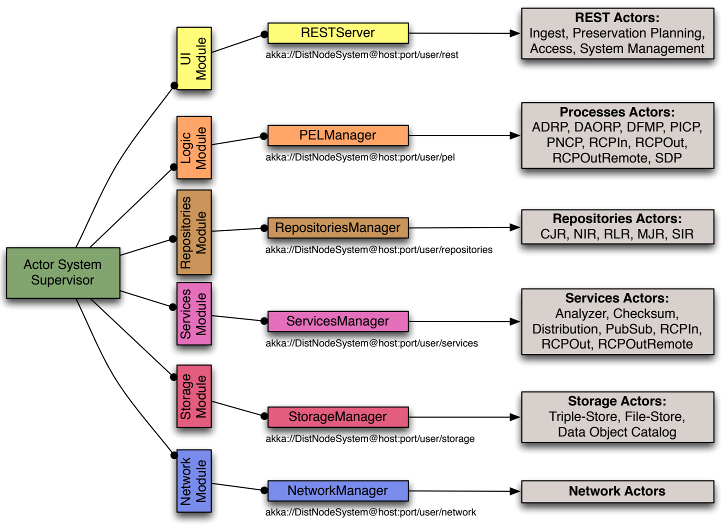
UI Module
RESTServer
akka://DistNodeSystem@host:port/user/rest
REST Actors: Ingest, Preservation Planning, Access, System Management
Logic Module
PELManager
akka://DistNodeSystem@host:port/user/pel
Processes Actors: ADRP, DAORP, DFMP, PICP, PNCP, RCPIn, RCPOut, RCPOutRemote, SDP
Repositories Module
RepositoriesManager
akka://DistNodeSystem@host:port/user/repositories
Repositories Actors: CJR, NIR, RLR, MJR, SIR
Actor System Supervisor
Services Module
ServicesManager
akka://DistNodeSystem@host:port/user/services
Services Actors: Analyzer, Checksum, Distribution, PubSub, RCPIn, RCPOut, RCPOutRemote
Storage Module
StorageManager
akka://DistNodeSystem@host:port/user/storage
Storage Actors: Triple-Store, File-Store, Data Object Catalog
Network Module
NetworkManager
akka://DistNodeSystem@host:port/user/network
Network Actors

Figure 4.6: DISTARNET Node Actor System

from each other. Each actor representing a module is supervised by the System Level Supervisor, which allows to isolate the failure by allowing only parts of the system to be restarted if needed. Further more, all communication inside a module is also done via messaging, and each module actor takes on the role of a supervisor for actors created beneath him. This allows the restarting of the different actors providing the module's functionality in case of failure.

This architecture basically creates a locally distributed system of service providers, where additionally also each service execution is isolated so that in the case of failure only this service will crash, and not the service provider, or the whole DISTARNET node.

4.4.2 User Interaction Module

The *User Interaction Module* is implemented as a HTTP Server providing a RESTful API. This API allows to control and interact with the deployed DISTARNET system node.

For the implementation of the HTTP server, Netty is used in combination with Unfiltered. The REST functionality if provided through the implementation of the routes with Unfiltered, which respond to the different HTTP request received by the Netty server. The main functionality is provided as an interface between the HTTP re-

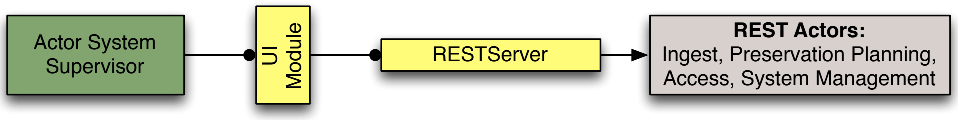

Figure 4.7: User Interaction Module Actor Hierarchy

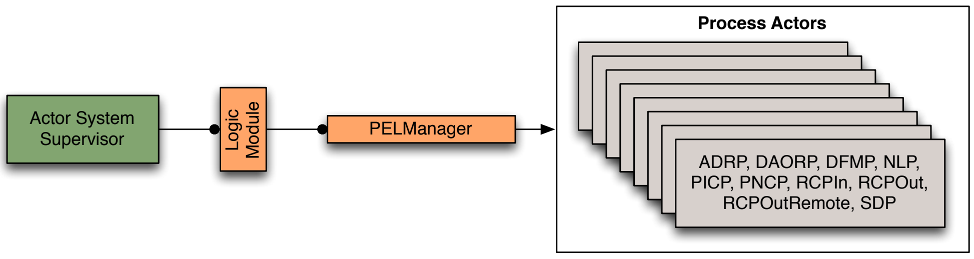
Actor System Supervisor
Logic Module
PELManager
Process Actors
ADRP, DAORP, DFMP, NLP, PICP, PNCP, RCPIn, RCPOut, RCPOutRemote, SDP

Figure 4.8: Digital Preservation Logic Module Actor Hierarchy

quests, and the DISTARNET modules, by translating the requests into *messages* which are then sent to actors in the different module, and serving the response back to the requester.

The actor hierarchy for this module is shown in Figure 4.7. Only the RESTServer was implemented in the prototype, as the REST actors themselves where not needed for the evaluation of the system.

4.4.3 Digital Preservation Logic Module

The Digital Preservation Logic Module is implemented as a series of actors organized in an actor hierarchy as shown in Figure 4.8. At the top of the hierarchy is the PELManager, which stands short for Process Execution Logic (PEL) Manager. The PELManager manages the execution of the DISTARNET processes, where each DISTARNET process is implemented as a distinct actor and is instantiated by the PELManager either timer based or on request as described in Section 3.4.2.

DISTARNET Process Execution

Following the discussion in Section 4.1.2 we here describe the process execution which will be provided by our own implementation of a Process Execution Logic.

The *Process Execution Logic (PEL) Manager* is responsible for the execution of the DISTARNET processes. The processes themselves will be implemented as *Finite State*

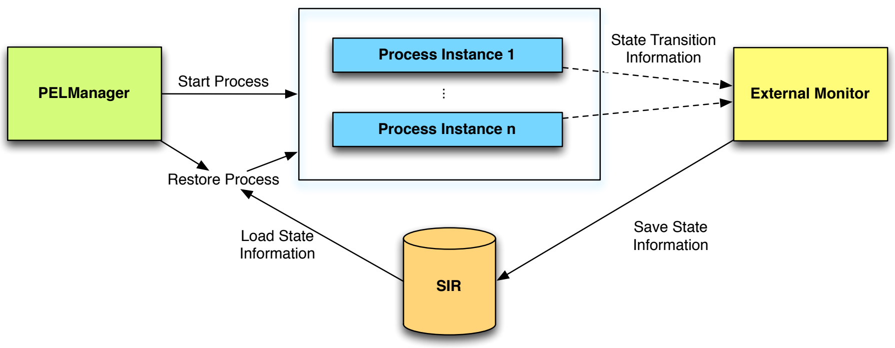

Figure 4.9: Process Execution Architecture Overview

Machines (FSM), where each state in the machine corresponds to one task in the process that needs to be executed.

The PELManager will initiate a process when triggered, by creating and starting a *Process Instance (PI)*. During the execution of a PI, an *External Monitor (EM)* will receive a message on every state transition, which will allow the persistence of the current state of a running process instance in the System Information Repository (SIR). Figure 4.9 shows an overview of the process execution architecture.

The creation of PI's will be triggered by a timer or a request as described in Chapter 3.4.2.

DISTARNET Process Implementation

The DISTARNET processes are implemented as *Finite State Machines (FSM)* by using the Akka FSM library. Each task in a DISTARNET process is implemented as a state. When all messages are exchanged, and a task is finished (e.g., send a message to execute service and receive service execution result), the FSM transitions to the next state.

We could have implemented the DISTARNET processes as a series of synchronous message calls. The problem with synchronous message calls is that they simply block the execution of an actor until a reply is received. This blocking also blocks the thread on which the actor is running. As mentioned earlier, in the Akka actor implementation, all actors share a defined number spawned system threads. If actor execution blocking is also done in some other actors, then the problem can arise that all available threads are blocked and that our system is standing still. The Akka FSM library

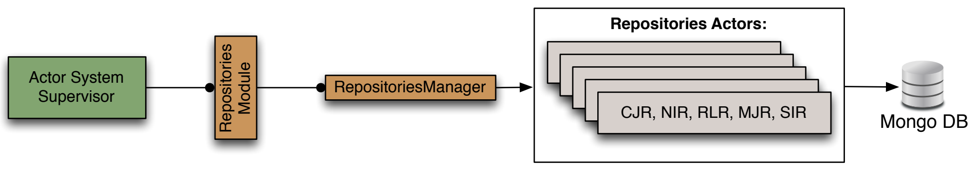

Figure 4.10: Repositories Module Actor Hierarchy

allows to implement efficiently serial execution in a system which is otherwise asynchronous, without blocking of any running threads.

4.4.4 Repositories Module

The Repositories Module houses the different repositories for the storage of information needed by the system. The RepositoriesManager manages the Node Information Repository (NIR), the Replica Location Repository (RLR), the Copy Job Repository (CJR), the Migration Job Repository (MJR), and the System Information Repository (SIR). Figure 4.10 shows the actor-hierarchy in this module.

The repositories are implemented as actors which store their corresponding information in separate Mongo DB collections. Each actor representing on repository, communicates through the Casbah API with the Mongo DB and is responsible for the management of the data stored inside its collection.

System Information Repository

The *System Information Repository (SIR)* is used to store the current state of a running process instance. The External Process Monitor maintains an entry for each running process, where on each change of state, the new information is updated. When the process finishes its execution, then the PEL removes the entry. Table 4.4 show the attributes of an entry.

4.4.5 Services Module

Figure 4.11 shows the actor-hierarchy in the *Service Module*. Every basic-service described in Section 3.4.4 is implemented as an actor. The ServicesManager actor routes any incoming requests to the corresponding actor implementation instance. All actor

Attribute	Description
instance id	Unique identifier of the running instance (actor name)
state name	Name of the last state
state data	Data taken over to the last state
timestamp	Timestamp of last change

Table 4.4: System Information Repository Entry Attributes

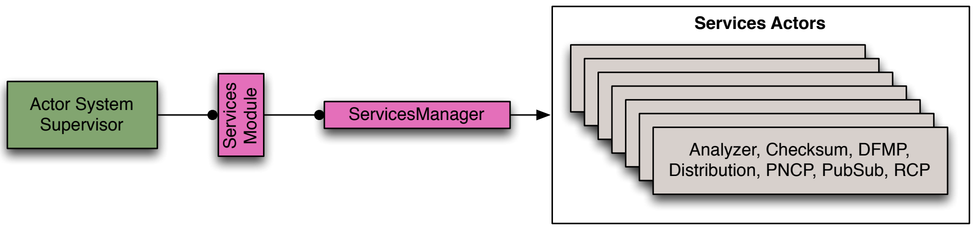

Figure 4.11: Services Module Actor Hierarchy

instances are instantiated and monitored by the ServicesManager when the node is started.

4.4.6 Network Module

The *Network Module* contains the *NetworkManager* actor as shown in Figure 4.12. The NetworkManager is responsible for the routing of initial messages to remote nodes. It implements a simple name to full address resolver, where messages internally addressed with DSN name and node name, are looked up in the DSN member registry, and forwarded accordingly.

In our implementation, the DSN member registry is a simple list loaded when the node is started, which contains the full addresses of every node in the DSN.

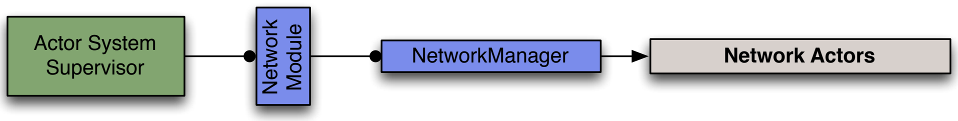

Figure 4.12: Network Module Actor Hierarchy

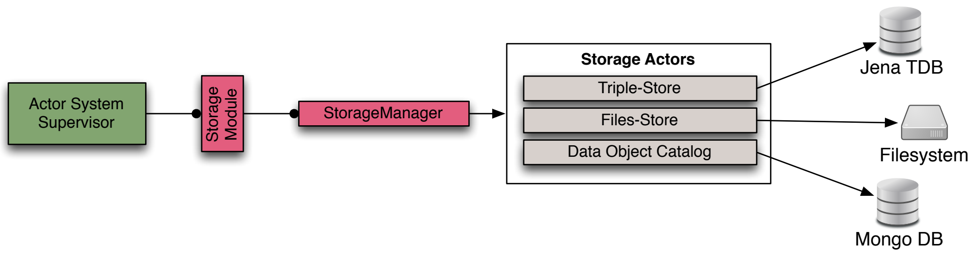

Figure 4.13: DAO Storage Module Actor Hierarchy

Attribute	**Description**
DSN	Unique identifier for the current DSN
node	Unique node identifier of the node storing this data object
data-uri	The URI used inside a DAO to identify the bitstream data for which this entry is meant
file-path	The the path to the storage location on the filesystem
checksum	The checksum of the bitstream

Table 4.5: Catalog Entry Attributes

4.4.7 DAO Storage Module

The *Storage Module* implements the features described in Section 3.4.6 and is shown in Figure 4.13.

The StorageManager instantiates and monitors every actor at node startup. The Triple-Store actor (named DAO DB-Store in the system model) uses the Jena API to manage and store the Distarnet Archival Objects (DAOs) which are represented as RDF graphs, in the Jena TDB triple-store. Next we have the File-Store actor (named DAO File-Store in the system model), mirrors the data stored in the triple store by storing them in RDF-XML representation on the filesystem. Finally, the Data Object Catalog actor, stores the bitstream data of the DAOs to the filesystem, and uses Casbah API to communicate with a Mongo DB instance, which is used to store persistently the different catalog information shown in Table 4.5.

4.5 DISTARNET Data Model Implementation

The Data Model DISTARNET implements supports the representation of *rich information networks* where the complex digital objects consisting of data and/or metadata constitute the *nodes* in the network, and the ontology-based relationships among these digital objects constitute the *edges*.

The integration of content management and the Semantic Web is motivated by the requirements imposed by the archiving community. A familiar example of this requirement would be the need to express well-known management relationships among digital resources such as the organization of items in a collection/subcollection, or creating arbitrary links between digital objects.

There are certainly a number of other schemes for the representation of these relationships such as conventional relational databases and formalisms like conceptual graphs. We believe that the need for extensible open-source solutions for representation, manipulation and the querying of these knowledge networks is best served by the products of the Semantic Web initiatives such as RDF[12], RDFS[13], and the Jena TDB triple-store[14].

The design of the DISTARNET (2.0) object model is a further development of the data model defined in the first implementation of DISTARNET (1.0) [80], where the model supported simple objects (e.g., a single image) but RDF was used in the background to store technical information about the system (e.g., locations of other replicas, IP addresses of neighbor nodes, etc.). In DISTARNET (2.0), the object model will be expanded by using RDF which will be used to represent every aspect of an object.

DISTARNET Model for Complex Objects

The DISTARNET object model provides support for different kinds of complex objects like documents, images, electronic books, and other compound information entities. Further, DISTARNET allows any combination of media types to be aggregated into complex objects. Additionally, by providing the ability to define assertions of relationships among objects, allows the creation of arbitrary links between DISTAR-

[12]`http://www.w3.org/TR/rdf-primer/`
[13]`http://www.w3.org/TR/rdf-schema/`
[14]`http://jena.apache.org/`

NET objects, managed collections/subcollections where a set of related DISTARNET objects can be represented as such, or even broader a set of objects that share some common characteristics (defined by semantic relationships).

To better understand the DISTARNET object model, we will take a look at the object model from two different perspectives: (1) the *relationship and representation* perspective, a simplified view where the internal structure of an object is not visible, but the interconnections to other objects and their meaning can be observed, and (2) the *inner* perspective, where the underlining structure of the objects is revealed.

We will begin by taking a look at a simplified view where the internal structure of an object is not visible, but the interconnections to other objects and their meaning can be observed. Afterwords we will go into more detail and discuss the RDF schema behind the data model, why we use RDF, and how we can calculate checksums for a RDF graph.

4.5.1 Relationships and Representations

In the relationship perspective, we can see that every DISTARNET Archival Object (DAO) can have one or more representations and that each object can have one or more relations to one or more other objects. This corresponds to the Logical Data Model we described in Section 3.6.1 but represented as a graph. As an example for an object with multiple representations we can look at an image or a document, where the content of both can be available in multiple formats. Every object and all its representations are identified through *Uniform Resource Identifiers* (URIs). The URIs follow the *Linked Data* paradigm. Linked Data[15] is a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF.

Figure 4.14 depicts three interrelated DISTARNET Archival Objects. In this directed graph, the darker nodes are DAOs, and the lighter nodes are the different representations that exist for the DAOs. We can observe two types of arcs that connect the depicted objects, the *relationship* arcs connect digital objects, and *representation* arcs connect digital objects to their respective representations. We can express this graph as RDF triples, which we can store in a triple-store, and the run queries upon.

In the diagram, each DAO has at least one representation, which is related by a

[15]W3C Linked Data: `http://www.w3.org/standards/semanticweb/data`

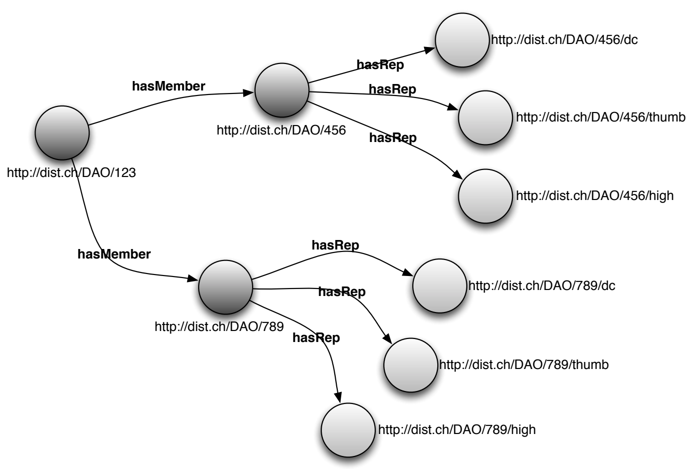
http://dist.ch/DAO/123
hasMember
http://dist.ch/DAO/456
hasRep
http://dist.ch/DAO/456/dc
hasRep
http://dist.ch/DAO/456/thumb
hasRep
http://dist.ch/DAO/456/high
hasMember
http://dist.ch/DAO/789
hasRep
http://dist.ch/DAO/789/dc
hasRep
http://dist.ch/DAO/789/thumb
hasRep
http://dist.ch/DAO/789/high

Figure 4.14: DISTARNET Object Relationships View

"hasRep" arc to the originating object. As an example, the node `http://dist.ch/DIOC/456` is an image object with three representations:

- Dublin Core record, `http://dist.ch/DO/456/dc`
- Thumbnail image, `http://dist.ch/DO/456/thumb`
- High-resolution image, `http://dist.ch/DO/456/high`

In Figure 4.14 we can also see an example of an inter-object relationship, represented by the "hasMember" arcs. The inverse "isMemberOf" relationships are not depicted for simplification, but are there. Here, the node `http://dist.ch/DIO/123` is a *Collection* consisting of two items, the nodes `http://dist.ch/DO/456` and `http://dist.ch/DO/789`.

4.5.2 RDF Schema

The *relationship and representation* perspective have provided us with a simple, access-oriented view of the digital resources and collections. The *inner* perspective will now provide a view of the underlying core data model for DISTARNET. The implementation of the core data model which represents the DISTARNET Archival Object is provided by the defined RDF Schema shown in Figure 4.15 where the different defined resource subclasses are depicted. There are two main resource subclasses. The first, the DAO subclass contains all the different DAOs that can be described by the

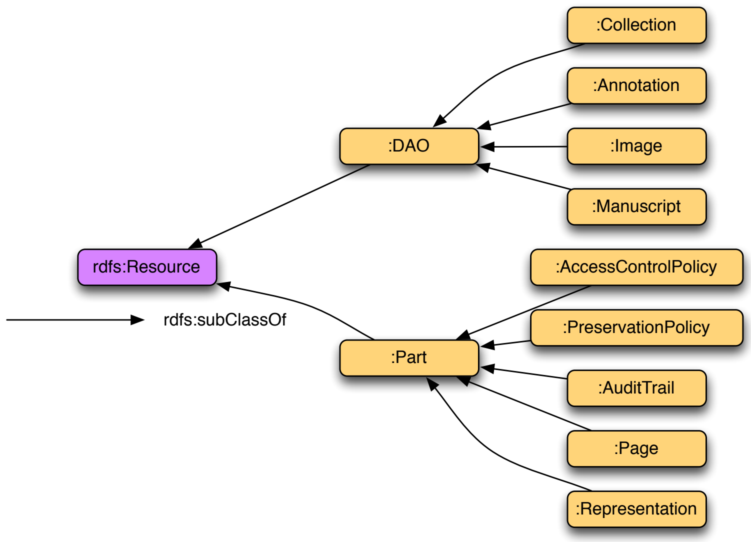

Figure 4.15: DAO RDF Schema - Resource Subclasses

data model. The second, the Part subclass contains the different parts a DAO can be made of.

Additionally, each resource subclass can be further described through properties, depicted in Figure 4.16 showing the RDF Schema which describes the different property subclasses. There are also two main property subclasses. The first, the objectProperty is used describe the relationships between instances of the DAO and Part resource subclasses, and the second, the dataProperty is used to describe values attached to instances of the DAO or Part subclasses.

4.5.3 RDF Model Checksum

As defined in 4.5 the DAOs are stored in RDF/XML files and loaded in the RDF triple-store. Consistency checks thus need to encompass both the RDF/XML files and the graph in the triple-store. When a DAO is checked, then following consistency checks need to be performed: (1) checking the consistency of the binary files that are archived (e.g., TIFF images, etc.), (2) checking the consistency of the RDF/XML files, (3) checking the consistency of the RDF graph of the DAO, and (4) check if the serialization of the RDF graph which is stored in the RDF/XML file is identical to the

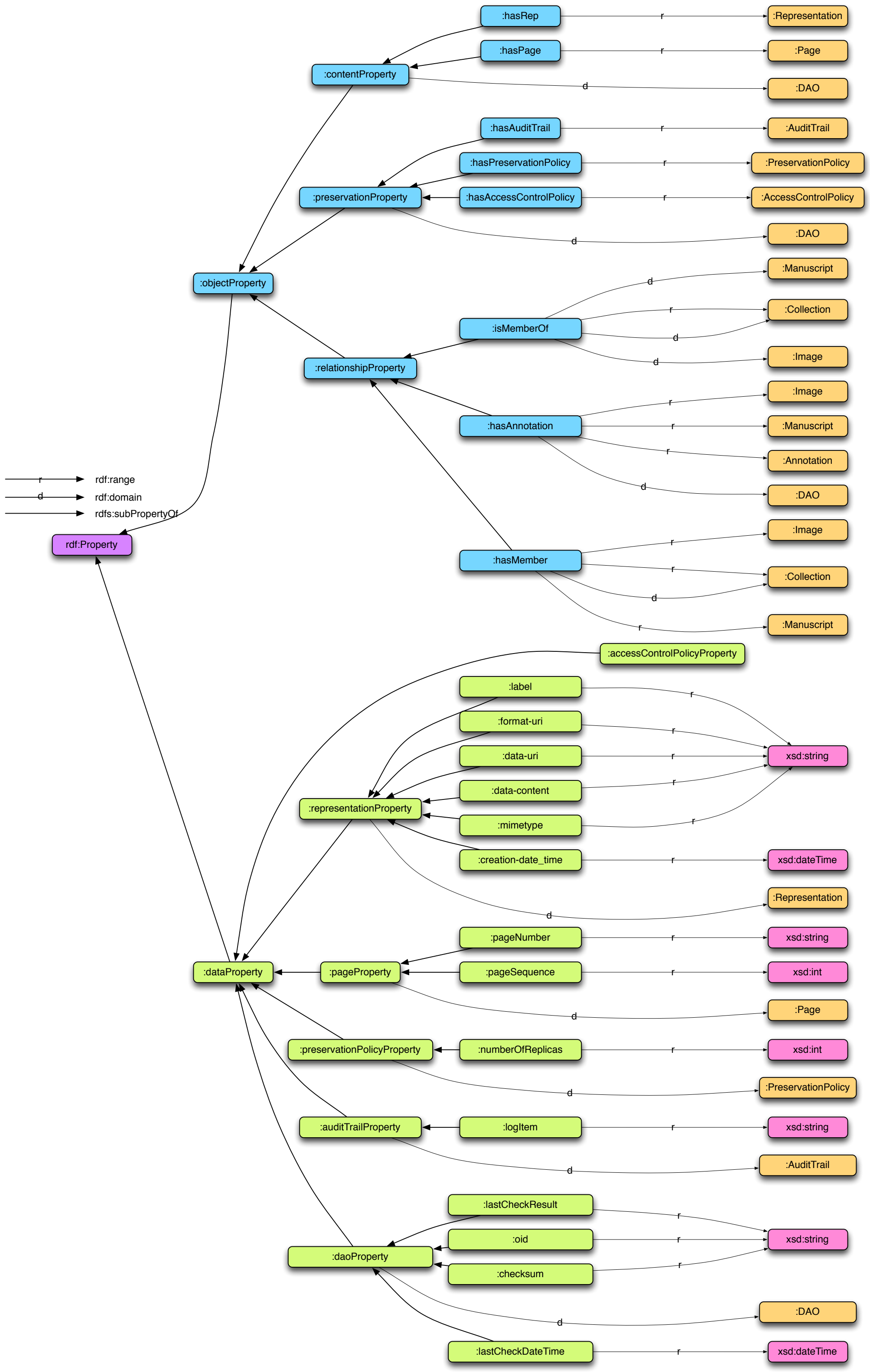

Figure 4.16: DAO RDF Schema - Property Subclasses

RDF graph stored in the triple-store.

To support these checks, three kinds of checksums are calculated: (1) the checksums of the binary files that are archived (e.g, tiff images, etc.), (2) the checksum of the RDF/XML file and (3) the checksum of the RDF graph of the DAO in the triple-store. To be able to compare the checksum of the graph stored in the triple-store and the checksum of the graph serialized in the RDF/XML file, we will calculate the checksum by using a set hash [88] . Basically what we will do is to calculate the checksums of the individual RDF statements and combine them.

4.5.4 Why RDF-based DISTARNET Archival Objects?

Motivation for using RDF: High flexibility and easy expendability to accommodate future needs. The combination of representing explicit relationships as RDF of a digital object and then mapping them to a triple-store offers the "best of both worlds". The explicit representation provides the basis for exporting, transporting, and archiving of the digital objects with their asserted relationships to other objects. The mapping to a triple-store provides a graph-based index of an entire repository and the basis for high-performance queries over their relationships. An added advantage of the dual representation is that the entire triple-store can be rebuild by importing and parsing the RDF-based information objects.

4.6 Summary

In this chapter, we have provided a further discussion of the requirements which have a direct implication for the implementation. These requirements range from node-engine fault-tolerance, over maintainability and extendability, to long-term use of the system.

Further we have described the concepts, frameworks and libraries that are used in the prototype implementation.

Following, we have provided an overview of the system architecture, a more detailed discussion of the different modules and how these are implemented, and the description of the data model implementation.

In the next chapter will now discuss the evaluation of the implemented prototype system, and the employed concepts.

5 Evaluation

In Chapter 3, we have presented a system model for long-term digital preservation of digital data, where we in general terms describe the DISTARNET processes that are needed to guarantee the integrity and consistency of the archived objects over the long-term. Following this general description, we have presented in Chapter 4 one possible implementation by employing these concepts. In this chapter, we now evaluate the implementation and through extension the concepts behind the implementation, with the use of realistic scenarios. The goal of the evaluation is to answer the questions if the implementation and the concepts behind, are working when applied to realistic scenarios, and what possible constraints the presented solution might have when applied to these scenarios.

The evaluation consists of two parts. In the first part, in Section 5.1, we define and employ a benchmark geared towards triple-stores as the DISTARNET data model implementation is based on RDF, in conjunction with the developed DISTARNET processes. We evaluate the feasibility and the constraints of using triple-stores for RDF-based metadata storage and management. In the second part, in Section 5.2, we perform a qualitative and quantitative evaluation of the DISTARNET system. The former looking at the correct execution of the DISTARNET processes, and the later looking at the performance of the system regarding the overall archiving storage capacity.

Global vs. Local Evaluation Viewpoint

When doing an evaluation of a distributed system such as DISTARNET, we can look at the system from a *global* or a *local* viewpoint. In the global view, we would evaluate the whole system, and look at how all nodes that form the system perform together. In the local view, we would isolate and evaluate one node or one part of the whole system.

During the course of this chapter, we take both approaches. For the evaluation of the performance of triple-stores in Section 5.1, which is just one but an integral part of our node, we assume a local viewpoint, but at the same time need to take the interactions with other nodes into account. In Section 5.2 we assume a global viewpoint so that the evaluation results mirror the functionality and performance of the system as a whole.

5.1 Evaluation of Metadata Management

In a distributed long-term digital preservation system, like DISTARNET, to guarantee the integrity and consistency, regular checks are applied to the archived objects, or in DISTARNET terms to the DISTARNET Archival Objects (DAOs). These regular checks are performed with the goal to timely detect inconsistencies, and trigger repair actions (e.g., to re-install a corrupted replica from another, uncorrupted one). The consistency checks generate a non-negligible additional load to the system which forms the special characteristics of long-term digital preservation systems, compared to other RDF database applications. Consistency checks and corrective actions will be referred to as *system processes*. In addition, read and write requests (*user processes*) need to be also considered in an archive.

There are a number of different benchmarks for evaluating the performance of triple-stores: the Berlin SPARQL benchmark (BSBM) [89], the Barton Library benchmark [90], the Lehigh University Benchmark (LUBM) [91], which focuses on inference and reasoning capabilities of RDF engines, and the SPARQL Performance Benchmark (SP^{2}Bench) [92].

All these benchmarks have in common that they do not, or only partly, address the specific access characteristics that can be found in a long-term digital preservation system.

The e-commerce scenario used by BSBM takes into account a similar access pattern a user would also have in an archiving system (*user processes*), e.g., it emulates the search and navigation pattern of a consumer looking for a product which can be

OpenLink Virtuoso v6.1	explicit: 15.4 billion URL: `http://www.openlinksw.com/`
BigOWLIM	explicit: 12.03 billion, implicit: 8.43 billion, total: 20+ billion URL: `http://www.ontotext.com/owlim/`
AllegroGraph	explicit: 20+ billion URL: `http://www.franz.com/`
4store	explicit: 15 billion URL: `http://4store.org/`
Bigdata(R)	explicit: 12.7 billion URL: `http://www.bigdata.com/blog/`
Jena TDB	explicit: 12.7 billion URL: `http://jena.apache.org`
Jena SDB	explicit: 650 million URL: `http://jena.apache.org`
Mulgara	explicit: 500 million URL: `http://www.mulgara.org/`
3store	explicit: 100 million URL: `http://threestore.sourceforge.net/`
Sesame	explicit: 70 million URL: `http://www.openrdf.org/`

Table 5.1: Triple-Store Capacities

compared to a user accessing an archive. However, it lacks support for *system processes* which make the particular semantics of a preservation system (e.g., consistency checking, data migration, etc.) and the recovery needed for corrupted data. Similarly, the Barton Library benchmark focuses on user access without special consideration of the access pattern imposed by system processes. The same also holds for LUBM and SP^{2}Bench. The benchmarks and therein covered problem areas of LUBM and SP^2 Bench while also important in the context of metadata management in a preservation context can not be used for performance measurements of triple-store-based metadata management.

Although RDF based data model and triple-stores as storage for metadata are used (at least partially) by some long-term preservation projects, there is no benchmark for evaluation of triple-stores that incorporate the access characteristics, data model and volume, of a long term preservation system. To close this gap, we have developed a benchmark.

There are also some general performance measures published for some specific triple-stores. Table 5.1 list capacity capabilities (number of triples) of different triple-store implementations[1]. These numbers, while interesting as they provide general pointers regarding the size of the data storage capabilities, are of static nature. They do not provide us with tangible information, regarding the way a particular triple-store will perform under such loads, when paired with specific access patterns in which we are interested.

In the following, we will thus introduce a novel benchmark for RDF-based meta-data management that jointly takes into account specific access patterns of long-term preservation systems that stem from both system and user processes, as such currently not present in any existing benchmark. Furthermore, we present the results of this benchmark applied to our distributed long-term digital preservation system DISTARNET in two realistic archiving settings, an Image Archive, and a Manuscript Archive.

5.1.1 DISTARNET Triple-Store Performance Evaluation

Based on the data model implementation discussed in Section 4.5, we specify two different usage scenarios that we then use in conjunction with a RDF/XML generator to create the test data. This test data is then been used for performance testing of the Jena TDB triple-store.

The goal of the benchmark is to test the performance of the triple-store subsystem under realistic usage patterns. We create a series of benchmarking queries that will allow us to test the performance of the triple-store with our data model under load, by taking into account the usage patterns of both system and user processes. The results will provide us with information about the limits of the triple-store under test, like what is the maximum number of triples that we can store in our triple-store while still having acceptable query speeds. It should further provide us with insights into possible bottlenecks of not only the triple-store, but also of our data model.

As one of the targeted user groups for the DISTARNET system are small institutions that do not have a large IT environment, we thus perform the evaluation on commodity hardware.

[1]The numbers were taken from `http://esw.w3.org/LargeTripleStores`

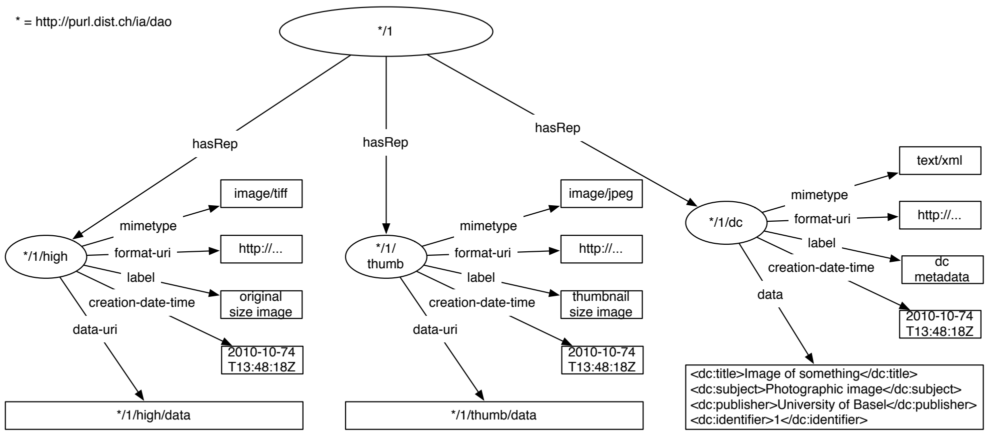

Figure 5.1: Image DISTARNET Archival Object Graph

5.1.1.1 Scenarios

To cover a wide usage spectrum, the benchmark will be performed using data generated for two different scenarios.

Scenario 1: Image Archive

The *Image Archive* consists of collections of images where each image has three representations: a TIFF and a JPEG representation, and a textual representation of the image in Dublin Core. Each representation is characterized by attributes such as mime-type, label, or creation-date-time. The TIFF and JPEG representations will only contain URIs to where the TIFF or JPEG files are stored.

Figure 5.1 depicts the Image DAO graph that describes the metadata graph that will be stored in the triple-store. Note that this is not a complete illustration of the whole graph but rather an excerpt that visually represents the differences between the image and the manuscript graph (scenario 2).

This scenario is motivated by its generality since the images can be exchanged for any data type (e.g., documents, audio/video, etc.) that can have multiple representations that need to be jointly archived.

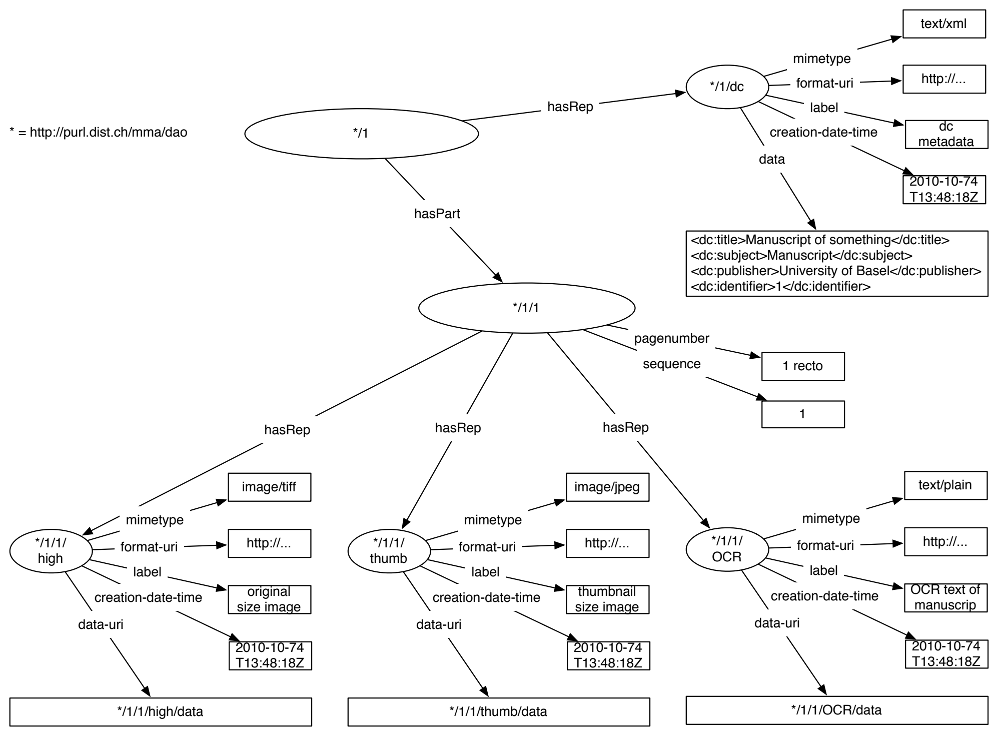

Figure 5.2: Manuscript Archival Object Graph

Scenario 2: Manuscript Archive

This scenario addresses collections of digitized manuscripts. Each manuscript consists of several pages, each one having multiple representations. Additional metadata describes the manuscript as a whole. For our scenario, we set the number of pages per manuscript to 100. Every page has three representations, namely TIFF, JPEG, and OCR Text. The metadata describing the whole manuscript consists of Dublin Core formatted text. Figure 5.1.1.1 depicts the Manuscript DAO graph.

This scenario is motivated by the SALSAH [93] (System for Annotation and Linkage of Sources in Arts and Humanities) project, a Virtual Research Environment[94], which allows collaborative work on medieval manuscripts, where a similar data model is used for the representation of medieval manuscripts in the system.

Scenario	Factor	# of Objects	Coll. Size	TS Data	# of Triples
1	10	1’000	100 GB	201 MB	29’020
1	100	10’000	1 TB	204 MB	290’200
1	1’000	100’000	10 TB	450 MB	2’902’000
1	10’000	1’000’000	100 TB	2.9 GB	29’020’000
2	10	1’000	10 TB	425 MB	2’515’020
2	100	10’000	100 TB	2.6 GB	25’150’200
2	1’000	100’000	1 PB	23.9 GB	251’502’000
2	10’000	1’000’000	10 PB	250 GB	2’515’020’000

Table 5.2: S1 and S2 Data Characteristics

5.1.1.2 Scaling Factor

The *RDF generator* creates the metadata representing the archives of different sizes for both scenarios. For this, we have defined a scaling factor F that specifies a multiple of a collection containing 100 archived objects. In both scenarios, we will perform the benchmark runs for scaling factor sizes of 10, 100, 1’000 and 10’000. The generated metadata corresponds to archive sizes of 1’000, 10’000, 100’000 and 1’000’000 objects. If we take the conservative assumption of a total of 100 MB of digital data per image object (including all representations), this implies for Scenario 1 required disk spaces of 100GB, 1TB, 10TB and 100TB, respectively.

For Scenario 2, if we again assume 100 MB of digital data per image object (which will be needed for all representations of each of the 100 pages), then the total disk space amounts to 10 TB, 100 TB, 1 PB and 10 PB, respectively. This total disk space represents the size of the whole archive managed by the metadata store. Table 5.2 shows the different scenarios (Scenario), the scaling factor values (Factor), the number of image or manuscript objects (# of Objects), the needed storage space for the whole archive (Coll. Size), the size of the triple-store on disk (TS Data), and the total number of triples (# of Triples).

5.1.1.3 Benchmark Queries

The benchmark queries are derived from the (system and user) processes’ access to metadata. The goal is to evaluate the scalability characteristics – and possible limitations – of the metadata store with increasing number of objects. We assume that the benchmark load reflects the system and user processes that need to be considered in the course of one day (i.e., a number of system processes is supposed to run daily). The focus of this benchmark is to evaluate the triple-store performance for metadata management. Therefore, the processes that have been implemented in the benchmark contain only the parts that pertain to the interaction with the triple-store.

System Processes

- Periodic Integrity Checking Process

P1: For each DAO, in the triple-store, find out if it was checked in the last 24h. If not then check if the checksums of the objects are OK, and mark as checked. If a checksum is not OK, then lock and mark DAO as corrupt. The whole archive must be checked once a day. P1 has read-write access to the triple-store.

P2: For all DAOs marked as corrupt (result of P1 and P3), replace subgraph with healthy version from another system retrieved with P3. We assume a failure rate of 10% of all DAOs. These 10% will be detected together by P1 and P3. The assumption of a 10% failure rate is a very conservative one and corresponds to the worst case that we anticipate based on the discussion in the paragraph about *Hardware Failure Rates*. P2 will also have read-write access to the triple-store.

P3: Queries a local node will receive from other remote nodes on which P2 runs. It returns the subgraphs and their checksums of a DAO identified by URI. Before sending the subgraphs, checksums are again calculated to check whether the DAO is OK. If not OK, then DAO is locked and marked as corrupt.
This query corresponds to the queries our node will receive from other nodes on which P2 would run. We assume that at least three copies per object are stored in the network. Therefore, the other two remote nodes can run this query against the local node in consideration (see also 5). Hence, we conservatively assume two concurrent requests. P3 has read-write access to the triple-store.

- Data Format Migration Process

  P4: Addresses data format migration, i.e., the actions to bring preserved digital content up-to-date when a new data format is available, or an existing format is deprecated. P4 converts each DAOs TIFF representation to JPEG2000. This process must finish within 90 days (for the complete collection) – so the whole collection could in principle be migrated four times a year, which is a very conservative assumption. P4 implements read-write access to the triple-store.

- Maintenance Process

  P5: Encapsulates the corrective actions needed to repair and maintain digital objects. For each affected subgraph of a DAO, the changed subgraph is retrieved, and the checksum of the subgraph is recalculated and updated after P4, P8, P9, and P10. This process accesses the triple-store in read-write mode.

User Processes

- Simple Read Queries

  P6: Query and return a certain number of objects belonging to a collection. The number of objects to be returned is determined by a normally distributed random variable. The number of process instances are calculated by multiplying 1/10 with the number of users and the scaling factor. P6 has read-only access.

- Complex Read Query

  P7: Find objects with creation date between two dates (randomly chosen from a list) that have a particular author, and some keywords that occur in the annotations. The number of process instances is calculated by multiplying 1/10 with the scaling factor and the number of users. P7 has read-only access to the triple-store.

- Write queries

P8:	User-created collection with some (randomly) selected objects. We conservatively assume that each user creates one collection per week (i.e., 1/7 per day). P8 needs read-write access to the triple-store.
P9:	Creation of an annotation. We assume that every user creates one annotation per day. Thus, P9 requires read-write access to the triple-store.
P10:	Represents the user creating a link between two DAOs. We assume that each user creates two links per day. P10 has read-write access.

Hardware Failure Rates According to [95] the failure rate of hard-disks is greater than what can be expected when looking at the MTTF numbers given by manufacturers. In their paper, they have analyzed data provided by different organizations deploying supercomputers and cluster systems. The hard-drives used in those server systems had a manufacturer specified MTTF between 1'000'000 and 1'500'000 hours. This would suggest a nominal annual failure rate (AFR) of at most 0.88%. However their findings where that in the field, the annual disk replacements rates typically exceed 1%, with 2-4% common and going up to 13% on some of the systems that where analyzed. These findings are also supported by [96] where they have analyzed the failure rates of the hard-drives deployed in the Google cluster. Their observed AFR was between 1.7% and 8.6%, and rose up to 10.5% when looking at drives with a high utilization rate.

Transactions

The queries Q8, Q9, and Q10 will be run by the users regularly and as such trigger Q5. This can present a problem for Q2, which will run daily since we have read and write queries that can occur on the same time on the same objects. Therefor we will need some form of transactions. For this, there are two possibilities. First, we can use Jena SDB which uses a relational database for storage. In this case, we could use the transaction capabilities of the relational database. In our case, we can not use SDB since it doesn't scale well if graphs with more than 100 million triples are stored (scenario 2 with factor 1000 has 211 million triples). For this reason, we will be using Jena TDB. But Jena TDB does not support transactions so our second possibility would be locking the model for exclusive read/write access.

Process-Type	Frequency	Concurrency	Access
P1	1	b * F	R/W
P2	0.1 * b * F	1	R/W
P3	0.1 * b * F	2	R
P4	1	1/90 * b * F	R/W
P5	P4 + P8 + P9 + P10	1	R/W
P6	1/10	u * F	R
P7	1/10	u * F	R
P8	1/7	u * F	R/W
P9	1	u * F	R/W
P10	2	u * F	R/W

Table 5.3: Query Frequency, Concurrency, and Access Type

5.1.1.4 Benchmark Mix

Table 5.3 shows the frequency and concurrency per process type that are used for creating the benchmark mix. The frequency denotes how often a specific process type will run, and the concurrency denotes how many possibly concurrent instances will be created each time the process runs. Multiplying the frequency and the concurrency yields the total number of instances of each process type created. Each process type in turn is implemented by means of a number of SPARQL queries. The factor F represents the scaling factor that is used to generate the test data. Increasing the scaling factor linearly increases the size of the metadata representing the archive, and thus run the benchmark on archives of bigger sizes, which in turn also increases the number of process instances of the process types that will run. The variable b is the base number of objects (e.g., image or manuscript) that are created for each F and variable u the number of users that access the system per F. The degree of concurrency of the user process types depend solely on u and F, and is the product of those two. Dependance on u is given as the users can access the system concurrently while the dependance on F represents the assumption that a larger archive is used by a larger number of users.

For the evaluation, we have set b to 100 (i.e., 100 image or manuscript objects are contained in a collection). The size of b is by itself not important, but rather the ratio of b and u as it defines the relation between system and user processes. For the bench-

Process-Type	Frequency	Concurrency	Absolute	Relative
P1	1	100	100	38.43 %
P2	10	1	10	3.84 %
P3	10	2	20	7.69 %
P4	1	1	1	0.43 %
P5	63	1	63	24.25 %
				74.64 %
P6	2	1	2	0.77 %
P7	2	1	2	0.77 %
P8	2	1	2	0.77 %
P9	20	1	20	7.69 %
P10	40	1	40	15.37 %
				25.36 %
TOTAL	151	110	260	**100.00%**

Table 5.4: Absolute and Relative Number of Process Instances

mark, we set the ratio between system and user processes to approx. 1/4 user and 3/4 system processes which reflects the special characteristics of a digital long-term preservation system. The amount of 1/4 of user processes also corresponds approx. to the ratio between the working hours of a workweek (40 hours) and the whole week (168 hours). Table 5.4 contains relative and absolute numbers of process instances for the base case with $F = 1$, $b = 100$ and $u = 20$.

5.1.1.5 Benchmark Implementation

The benchmark is implemented in Scala and can be run from any current JVM. The benchmark test driver application takes as parameter the location of the SPARQL endpoint. For the communication to the triple-store, the SPARQL 1.1 HTTP protocol[2] is used. The system and user processes implement their operations on the RDF graphs by using SPARQL 1.1 Query[3] and Update[4]. The benchmark mix is implemented by serializing the system and user processes described in Table 5.4, depending on the size of F which is given by the number of DAOs found in the triple-store. This guarantees at runtime an evenly distributed access pattern to the triple-store,

[2]http://www.w3.org/TR/2011/WD-sparql11-http-rdf-update-20110512/
[3]http://www.w3.org/TR/2011/WD-sparql11-query-20110512/
[4]http://www.w3.org/TR/2011/WD-sparql11-update-20110512/

where the system and user processes are evenly intermixed and spread over the whole duration of the benchmark.

The benchmark can be run on any triple-store that exposes a SPARQL endpoint with enabled update operations, and into which the RDF data from the data generator is loaded.

5.1.2 Benchmark Evaluation Results

5.1.2.1 Evaluation Setup

Since one of the targeted types of organizations for DISTARNET are small archives that do not possess a large IT environment, we decided to perform the evaluation on commodity hardware. The evaluation was performed on an Apple Mac Pro with 2 x 3GHz Dual-Core Intel Xeon CPUs, 24GB RAM and a dedicated 2TB SATA disk drive. The triple-store used in the evaluation was Jena TDB[5], a natively implemented triple-store, with a SPARQL endpoint provided by the Fuseki server[6]. The benchmarking software implementing the benchmark mix with the calls to the SPARQL endpoint was run from the same machine as the triple-store.

5.1.2.2 Bulk Load Times

The Jena TDB triple-store has two command line utilities for fast initial loading (i.e., build TDB indexes) of RDF data into an empty TDB store, namely `tdbloader` and `tdbloader2`. They differ in the way indexes are generated. `tdbloader` builds the node table and the primary indexes first. After that, it builds the secondary indexes. `tdbloader` can also be used for incremental loads, i.e., to load data into an existing TDB database. `tdbloader2`, in contrast, builds just the node table and text files for the input data using Node IDs. It then uses UNIX `sort` to sort the text files and produce text files ready to be streamed into `BPlusTreeRewriter` to generate B^+Tree indexes. `tdbloader2` cannot be used to incrementally load new data into an existing TDB database, but it is faster when working with larger data sets[7].

For the two scenarios, we have used the `tdbloader2` command. Table 5.5 contains the load time durations and the rate in triples per second (TPS) for the two scenarios

[5]http://openjena.org/TDB/
[6]http://openjena.org/wiki/Fuseki
[7]http://seaborne.blogspot.com/2010/12/performance-benchmarks-for-tdb-loader.html

Scenario	Factor	# of DAOs	# of Triples	TS Size	Duration	TPS
1	100	1′000	29′000	0.2 GB	00m 09s	3′224
1	100	10′000	290′200	0.2 GB	00m 42s	6′910
1	1′000	100′000	2′902′000	0.5 GB	06m 59s	6′926
1	10′000	1′000′000	29′020′000	2.9 GB	01h 07m	7′206
2	10	1′000	2′515′020	0.4 GB	06m 32s	6′416
2	100	10′000	25′150′200	2.6 GB	01h 14m	5′608
2	1′000	100′000	251′502′000	23.9 GB	13h 38m	5′120
2	10′000	1′000′000	2′515′020′000	250.0 GB	97h 20m	7′177

Table 5.5: Bulk Load Times for Scenario 1 and 2

and different scaling factors. This table shows that there is a substantial amount of data that need to be managed, even though we are only dealing with the metadata of the archival system.

5.1.2.3 S1 and S2 Evaluation Results

In the following, we will discuss the results of our evaluation runs made with scaling factor F set to 10, 100, 1′000 and 10′000 for both scenarios (summarized in Table 5.6). Figure 5.3 depicts the durations of the evaluation runs divided by the scale factor. The change in duration from $F = 10$ to $F = 100$ is sub-linear but can be explained by the short run time for $F = 10$ and the overhead of the program for running the evaluation. The change in duration is rising slightly between $F = 100$ and $F = 1'000$ for both scenarios, and then between $F = 1'000$ and $F = 10'000$, S1 stays almost linear while S2 doubles in duration. This doubling can be explained by the high load, i.e., 2.5 bn triple and adding 11m new ones, combined with the limitations for executing parallel write operations.

5.1.2.4 Discussion

The benchmark evaluation times for both scenarios and all F values are well below the 24h mark except for $S2$ with $F = 10'000$ where the duration is over 48h. This means that the evaluated triple-store can be used for the storage and management of metadata for DISTARNET archives of a size of approx. 5 PB (as this corresponds to the case where all benchmark queries terminate within the 24h margin).

Factor	**10**	**100**	**1’000**	**10’000**
S1				
New Triples	3’438	34’476	344’914	3’449’197
P-Type Instances	2’618	26’192	261’936	2’616’364
Duration	19s	2m 40s	46m 4s	7h 46m
S2				
New Triples	11’060	111’405	1’106’400	11’145’304
P-Type Instances	2’618	26’192	261’936	2’619’364
Duration	1m 37s	13m 28s	2h 35m	2d 10h 16m

Table 5.6: Evaluation Results for S1 and S2

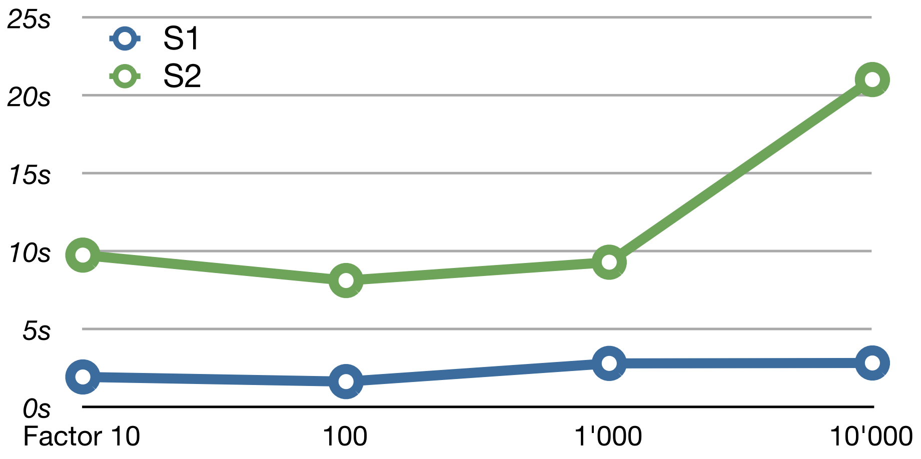

Figure 5.3: Evaluation Run Durations in Seconds per Scale Factor in S1 and S2

Parallelism

The implementation of the benchmark utilizes parallel execution of the process instances whenever possible. As described in Section 5.1.1.5, the execution is done in cycles where one cycle is equivalent to the benchmark mix for $F = 1$ and where all processes are run in a serial fashion. The different process instances of the same type are always run in parallel where possible. The process instances for P1, P4, P6 and P7 are all run in parallel. Presently, the degree of parallelization of the write processes is limited, due to the limitations of the underlying Jena TDB triple-store as only a multiple-reader / single writer locking model is supported.

As a consequence, the parallel execution of the process instances have a rather small positive impact during the execution of the benchmark. The Jena triple-store does not answer the request in parallel for P1 and P4 as these process types require write access during which Jena is locking the model for all other requests.

5.2 Evaluation of the DISTARNET System

In this section, we provide an evaluation of the DISTARNET system as a whole. The main focus of the evaluation is on the DISTARNET processes, i.e., their ability to cope with failures that the system can encounter, as described in Section 3.3, i.e., infrastructure and content faults.

The failures of the execution environment, i.e., node engine faults are not part of this evaluation. Although an important feature in a productive running system, the main focus of the implemented prototype, and this evaluation lies primarily on the validation of the main DISTARNET processes as they represent the most important and central elements of the system.

The evaluation is structured as follows. First we do a qualitative evaluation of the system, for which we use four test scenarios that involve the main processes running in the DISTARNET system. These scenarios cover single infrastructure and content faults, data format migration tasks, and a combination of all three.

Secondly, we conduct a quantitative evaluation, by using measurements of the execution times of a series of processes in combination with approximations, and through extrapolation provide an overall performance of the system. These results are then discussed in comparison to the results from Section 5.1.1.1.

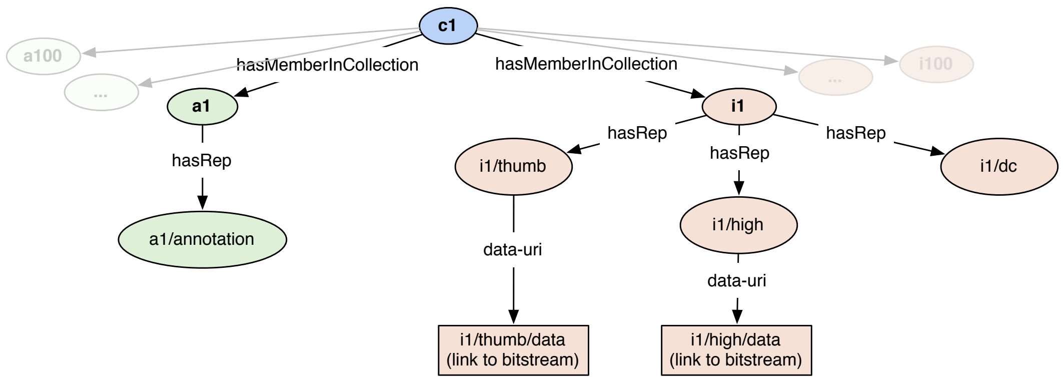

Figure 5.4: Evaluation Data Structure

5.2.1 Cooperating Image Archives Scenario

Both the qualitative and quantitative evaluation of the DISTARNET system use this scenario that represents a network of four cooperating Image Archives. In this scenario, every Image Archive deploys one node running the DISTARNET system software and uses the other nodes in the network for the storage of remote replicas. This setup is motivated by the scenario described in Section 1.1.2 regarding the National Museum of History and Native Art, where each institution has only the means to deploy one DISTARNET node, and through cooperation with other institutions has access to additional remote DISTARNET nodes, on which they can deploy their remote replicas. The here described four nodes create what we also call a DISTARNET network.

The data used in this scenario is an extended version of the data from Section 5.1.1.1. As before, the scenario represents an Image Archive, where each Image DAO is comprised of three representations, i.e., Dublin Core metadata, JPEG, and TIFF. The JPEG and TIFF representations have additionally references to randomly generated bitstream data. The DAOs are grouped together into collections, where one collection is comprised of 100 Image DAOs. Additionally, and this is what is different in comparison to the Image Archive Scenario used in Section 5.1.1.1, the collection also contains 100 Annotation DAOs. In summary, one collection thus contains 100 Image DAOs and 100 Annotation DAOs. We use synthetically generated data, consisting of one such collection, as our base data for the different evaluations. Depicted in Figure 5.4 is an overview of the data structure.

5.2.2 Qualitative System Evaluation

During the qualitative system evaluation, we put the network of Image Archives, described previously, through a series of test scenarios. In Scenario 1, we simulate the destruction of one node, in Scenario 2 we simulate the corruption of archived content, in Scenario 3 we simulate data format migration, and in Scenario 4 we combine all three together.

Evaluation Environment

The implementation of the test scenarios is comprised of a collection of multi-JVM test cases, running in the development environment. This allows to run a predefined number of DISTARNET nodes, and create a DISTARNET network on the same physical hardware while still running each node in its own JVM. The different test cases are used to send messages in an orderly fashion to the nodes running in the DISTARNET network and by doing so initiate certain effects, e.g., kill nodes, corrupt content, etc. Afterwards, subsequent test evaluate the reactions of the individual nodes, i.e., the result of these reactions, such as for example the generation of additional replicas, repairing of content, etc.

The execution of the test cases is synchronized between all nodes, which allows a precise control of their execution order throughout the network, and also allows to query the state of each node at key points in time during the execution of a test.

Running these tests in a multi-JVM environment, has shown to have a negative impact on the speed of the execution, as the limited system resources are divided between multiple threads of execution. Thus, this kind of execution can not be used for quantitative evaluation, as it would distort the results. However, for the purpose of this qualitative evaluation, we can use a mutli-JVM environment, as long as we restrict the number of the running nodes, to a number where each node has enough resources to provide a correct execution, i.e., the other nodes running concurrently are not starved for resources, and we are not interested in the execution time as the process execution will slow down. The evaluation is performed on an Apple Mac Pro with 2 x 3GHz Dual-Core Intel Xeon CPUs, with 24GB RAM, and a dedicated 2TB SATA disk drive. As we need to run at most four nodes during this qualitative evaluation, the four cores provided by our hardware are enough to ensure a correct execution during the evaluation.

Initial Network State

Before running each test scenario, we bring the network of Image Archive nodes to the same initial state. This state consists of four nodes being started (each representing an archive). Each node is ingested with its own collection of DAOs consisting of 100 image DAOs and 100 annotation DAOs, as described in Section 5.2.1. All four nodes belong to the same DISTARNET Sub-Network (DSN). The nodes are named NODE01 through NODE04. Each node creates for each DAO ingested on this node, two additional replicas on the next two nodes, e.g., NODE01 on NODE02 and NODE03, NODE02 on NODE03 and NODE04, NODE03 on NODE04 and NODE01, and NODE04 on NODE01 and NODE02.

The initial network state is created through the execution of the following steps:

1. Load data in the triple-store on every node, which represents the 100 image DAOs and the 100 annotation DAOs.
2. Create the catalog entries and data on every node, representing ingest of 100 image DAOs.
3. Create the appropriate Replica Location Repository (RLR) entries for our test data (that have been created in the previous steps) on each node.
4. Create the initial Node Information Repository (NIR) entries on each node.
5. Initiate the Automated Dynamic Replication Process (ADRP) on every node, which will create and execute Reliable Copy Process Out (RCPOut) jobs, which will create replicas on two other nodes (as described earlier).
6. Initiate the State Dissemination Process (SDP) on every node, so that the RLRs and NIRs across the network are updated with the newest information regarding the locations of all replicas throughout the network, and additional information about all nodes present in the DSN.

These steps are part of every test scenario, where they are executed at the beginning before any actual tests of a scenario are executed. Figure 5.5 depicts the created initial DISTARNET network state.

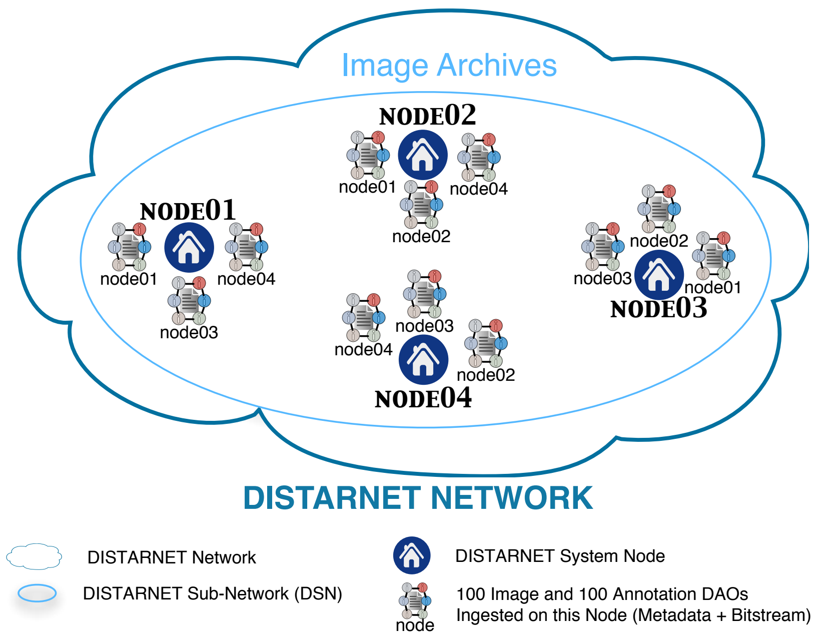

Figure 5.5: DISTARNET Network of Four Image Archives after Initialization

This initial network state represents the state of the DISTARNET network after all nodes have been ingested with the digital objects that need to be archived (steps 1. - 4.), and after additional replicas have been created and distributed across the archival network (steps 5. - 6.).

5.2.2.1 Test Scenario 1: Node Destruction

In the first test scenario, we simulate the use case, in which one of the nodes in the DISTARNET network of Image Archives has permanently gone offline (e.g., destructed through an earthquake, flood, etc.).

This scenario represents an extreme situation. Nevertheless, its occurrence needs to be countered promptly as this event endangers all the DAOs throughout the network, whose replicas were stored on the now destructed node, as the required overall redundancy prescribed by the preservation policies is not upheld anymore. The system needs to employ measure to bring the DISTARNET network back to a stable state. This stable state must provide the same redundancy of the stored content as it was before the occurrence of the node destruction.

To pass successfully this test scenario, the archive network needs to pass a series of test cases that are testing the system for the following behavioral sequence:

1. After a node in the network is killed, the other nodes in the network must detect through the Periodic Neighbor-Node Checking Process (PNCP) that a node is lost.

2. The next node in the responsibility chain must take over the responsibility for the DAOs ingested on the lost node.

3. The node taking the responsibility must initiate the creation of additional replicas in the network, to compensate for the lost replicas stored on the missing node.

To verify if the DISTARNET network under test is behaving as described, we use a series of test cases that verify the state of the network at key points in time, and is structured as follows:

1. Initialize the DISTARNET network under test to the initial state as described earlier.

2. Induce test scenario by shutting down NODE01.

3. The PNCP running on every node (NODE02, NODE03, and NODE04) detects that NODE01 is lost. After *Lost Node* detection occurs, the Node-Lost Process (NLP) should be started automatically on all nodes, after which NODE02 should have taken responsibility for the DAOs of NODE01 as it is the next node in the responsibility chain after NODE01 in our setup. Also, all nodes should have initiated ADRP to create additional replicas on other nodes, countering the disappearance of NODE01.

4. To verify that the network of nodes has reached a stable state, we check at the end a sample of RLR entries on every node, to see if there are again three replicas of each nodes DAOs in the DISTARNET network.

In this test scenario, we only induce one failure, from which the DISTARNET network automatically recovers from. The initiated recovery process can tolerate additional failures during recovery, which will be covered in the multi-failure test scenario in Section 5.2.2.4. The recovery from the loss of a node, and even multiple nodes at the same time, and the recreation of the lost replicas stored on these nodes is only restricted by the availability of other replicas of the same DAO in the DISTARNET network. If all replicas of a DAO are lost at the same time, then the recovery can not be performed. In the case that at least one replica of the DAOs is available, then the remaining nodes will be able to create additional replicas, and bring the DISTARNET network back to a stable state. The idempotent nature of the DISTARNET processes, allows them to be restarted in case of a failure during their execution, until a successful outcome, or a definable retry threshold is reached. In the case of the retry threshold, notifications would be dispatched, and external operator intervention would be needed.

5.2.2.2 Test Scenario 2: Content Corruption

Again as before, we use the network of Image Archives, and now we simulate in this scenario the use case, where on one node, parts of the archived content, i.e., the DAOs become corrupted.

This is a likely event, that will occur at a higher rate than the previously discussed scenario involving node destruction, especially in larger archives, as the number of for example, hard-disk failures rises with the number of deployed drives. When this event occurs it needs to be dealt with immediately as any subsequent content corruption on say other nodes, can lead in the worst case to a complete and permanent loss of the archived content, if as a result of content corruption all the replicas on local and remote nodes of the same DAO become unusable.

The DISTARNET network under test will need to present the following behavior, to pass successfully this test scenario:

1. Periodic Integrity Checking Process (PICP) should check all local DAOs and find corrupted content on the node on which the corruption has happened.
2. The node will then initiate the repairing of the corrupted DAOs by getting fresh copies of the missing content from other nodes in the network.

As before, we use a collection of test cases to verify the correct behavior. The collection of test cases will does the following:

1. Initialize the DISTARNET network under test to the initial state as described earlier.

2. Induce scenario by deleting the bitstream files on NODE01 of 10 different DAOs.

3. The PICP running on NODE01, should detect the corruption of the DAOs, and automatically initiate the DAO Repairing Process (DRP) for each detected corruption.

4. To verify that the DISTARNET network was restored to the state before the corruption occurred, we wait for DRP to finish, and for the execution of PICP. This time PICP should not find any corrupted DAOs.

5.2.2.3 Test Scenario 3: Data Format Obsolescence

In our third scenario, we look at the use case, of data format obsolescence. In this use case, the interpretability of some of the DAOs in the DISTARNET network is threatened by an obsolete data format. To counter this threat, we need to perform a data format migration task.

As we want to provide a long-term preservation solution with DISTARNET, the task of data format migration is very likely to happen in the future of every archive. As time goes by, the requirement of interpretability of the archived content will raise the need for data format migration as data formats become inevitably obsolete. As such, we need to provide the means to allow controlled changes to the archived content, and subsequently consistently propagate these changes throughout the network to all the remote replicas.

To pass successfully this test scenario, our DISTARNET network of Image Archives must provide the following behavior:

1. Execute migration job.

2. The additional representation created in the course of the migration job execution must be appended to the DAO for which the data format migration is performed.

3. The DAO's checksums (graph and bitstream) need to be updated.

4. The updated DAO (graph and bitstream) will need to be sent out to all remote nodes where the old version is stored.

As before, the correctness of behavior is verified through a collection of test cases, which contains the following steps:

1. Initialize the DISTARNET network under test to the initial state as described earlier.
2. Induce scenario by creating a migration job (e.g., containing instructions to create additional representations by converting TIFF bitstreams to J2K bitstreams) in NODE01, which should be detected on this node, and lead to the execution of the Data Format Migration Process (DFMP) on NODE01.
3. To verify if the DISTARNET network has achieved a consistent state after the migration job has finished, we run on NODE01, NODE02, and NODE03 a check against the Catalog, to see if the additional representation was created, and distributed to the other nodes storing replicas of the DAO for which the DFMP was run.
4. Additionally, PICP will run on every node, which should not find any corrupted content. This will test if the appending of the created representation, and the recalculation of the checksums was done correctly.

5.2.2.4 Test Scenario 4: Multi-Failure

In our last test scenario, we run the DISTARNET network of Image Archives through a simulation of a combination of infrastructure failure, content corruption, and data format migration task happening at the same time.

This scenario represents the worst case, which should not be very common, but is still a possibility, and needs to be dealt with. As in all the other previous scenarios, the goal is to have at the end of the scenario, a DISTARNET network of Image Archives, which remaining nodes reside in a stable and consistent state.

Our archival network needs to provide the following behavioral sequence to pass successfully this test scenario:

1. PNCP on all nodes should detect that NODE01 is lost, NODE02 should take over responsibility and create additional replicas.
2. PICP on NODE02 should detect DAO corruption and initiate DRP.

3. Execute migration job on NODE03.

The collection of test cases verifying the correct behavior contains the following steps:

1. Initialize the DISTARNET network under test to the initial state as described earlier.
2. Induce scenario by shutting down NODE01, deleting the bitstream files on NODE02, and creating a data format migration job on NODE03.
3. The PNCP should run on every node, and should find that node01 is lost. After *Lost Node* detection occurs, the Node-Lost Process (NLP) should be started on every node. As before, NODE02 should create an additional replicas for DAOs from NODE01, and also NODE03 and NODE04 should created additional replicas.
4. The PICP running on NODE02, should detect the corrupted DAOs, and repair them.
5. The presence of a DFM job on NODE03, leads to the execution of the DFMP, which should create an additional representation and update the DAOs on NODE02 and NODE03 which now store the remote replicas as NODE01 is down.
6. To verify that our network of Image Archives has reached a stable and consistent state, check a sample of RLR on every node, to see if there are again 3 replicas of each nodes DAOs in the network. Also, we check the Catalog on every node to see if the additional representation created by DFMP is present and was distributed around the network.
7. Additionally, PICP will run on every node, which should not find any corrupted content, verifying that our archival network is in a consistent state.

5.2.2.5 Qualitative Evaluation Summary

For the qualitative evaluation of the implemented DISTARNET system, we have defined a test setup that represents a cooperative network of four Image Archives, where each archive deploys a node running the DISTARNET system software. For the evaluation, we have used four scenarios, derived from real world use cases, which cover the core functionalities of the DISTARNET system.

The four scenarios cover failures ranging from infrastructure to content faults and are implemented as a collection of test cases. In the first scenario, we simulate the destruction of one node, in the second scenario we simulate the corruption of archived content, in the third scenario we simulate data format migration, and in the fourth scenario we combine all three together.

For each of the four scenarios, the system passes the execution of all the defined test cases without any errors. This error free execution lead us to the conclusion that the core functionality provided by the implementation of the DISTARNET system is working correctly. As the prototype implementation is closely based on the concepts described in Chapter 3, we further conclude that the therein described system model for a distributed long-term digital preservation is also validated.

5.2.3 Quantitative System Evaluation

In the following quantitative evaluation, we use the DISTARNET network scenario described in Section 5.2.1. This scenario is a good representative of the scenarios discussed in Section 1.1, which we will later on discuss further in the context of the results of the quantitative system evaluation.

The quantitative system evaluation we are performing is based on a combination of practical measurements in combination with analytical approximations. For the purpose of this evaluation, we thus measure the execution time of different processes running on one node, with the deployed DISTARNET prototype as used in the qualitative evaluation. We then combine this measure with approximations for inter-node transfer times, to attain a comprehensive measure, that represents a good approximation of a running DISTARNET network. Additionally, we extend these results with additional extrapolated data points.

The goal of this evaluation is to provide approximations regarding the quantitative performance of the system, i.e., how many TB of data can be managed on one node under the constraint that the main DISTARNET processes need to run and finish within 24 h, which will allow us to compare these results with the results obtained during the evaluation performed in Section 5.1.1. As this 24 h constraint represents a very conservative restriction, we also discuss more practical and relaxed alternatives,

Scaling Factor	**1**	**10**	**100**	**1’000**	**10’000**	**100’000**	**1’000’000**
Nr of Image DAOs	100	1 K	10 K	100 K	1 M	10 M	100 M
Overall Archive Size	10 GB	100 GB	1 TB	10 TB	100 TB	1 PB	10 PB

Table 5.7: Overall Data Collection Size per Scaling Factor

and how they affect the overall result.

5.2.3.1 Test Data

The data that we use during the evaluation of the DISTARNET archiving system is the same as described in Section 5.2.1 containing generated synthetic content. To provide an easier discussion, we assume that the overall size of an image DAO is 100 MB (i.e., just the size of the TIFF bitstream, neglecting the Dublin Core, and JPEG representations), and neglect the size of the annotation DAO. Taking this data collection as the base for our evaluation, i.e., scaling factor $F = 1$, we can derive through extrapolation the overall data collection size for each scaling factor we use in the evaluation. This data is presented in Table 5.7. As an example, for a scaling factor $F = 100$, we thus have 100 collections with each containing 100 image DAOs and 100 annotation DAOs. As we only count the image DAOs with 100 MB each, we totally have 100 times 100 image DOAs equaling to 10’000 image DAOs with an approximated overall data collection size of 1 TB.

5.2.3.2 Evaluation Procedure

The mix of DISTARNET processes we use in the evaluation corresponds to the system process benchmark queries and mix as described in Section 5.1.1.3 and 5.1.1.4.

For the evaluation we time the duration a DISTARNET node system needs to execute the process mix with a data scaling factor $F = 1$, and then extrapolate the theoretical system performance for the different scaling factors under a linear scaling assumption.

Process Mix

In the first step, we run the Periodic Integrity Checking Process (PICP) over the Image Archive data, where we beforehand deleted 10 % of the TIFF images corresponding

Scaling Factor	1	10	100	1’000	10’000	100’000	1’000’000
Image DAOs	100	1 K	10 K	100 K	1 M	10 M	100 M
Archive Size	10 GB	100 GB	1 TB	10 TB	100 TB	1 PB	10 PB
Duration w. 100 Mb/s	🟩 129.415 s	🟩 0.36 h	🟩 3.59 h	🟥 35.95 h	🟥 15 d	🟥 150 d	🟥 1’495 d
Duration w. 1 Gb/s	🟩 50.894 s	🟩 0.14 h	🟩 1.41 h	🟩 14.14 h	🟥 6 d	🟥 59 d	🟥 589 d

Table 5.8: Evaluation Results Overview

to a 10 % data loss rate. The PICP will detect the data loss, and start the DAO Repairing Process (DRP) to repair the data by using the Reliable Copy Process In (RCPIn) to recover fresh copies of the lost data. In the second step, the Data Format Migration Process (DFMP) will run for 1 DAO, converting a TIFF representation to a JPEG2000 representation, and using the Reliable Copy Process Out (RCPOut) to send the updated DAO with the newly created representation to two remote nodes.

Inter-Node Transfer Times

To attain a good approximation of a running DISTARNET network, we use approximations of the inter-node network transfer times, representing time that we need to add to the times previously measured, to compensate for time needed for network traffic. Using approximated values gives us one parameter more, which we can regulate, and provides us with additional data points for the discussion. For the approximation of the inter node transfer times, we use two network bandwidth assumptions. The first representing a 100 Mb/s network, with a theoretical transfer capability of 12.5 MB/s, where it theoretically takes 8 seconds to transfer 100 MB of data. The second representing a 1 Gb/s network between the nodes, with a theoretical capability of 125 MB/s, where it theoretically takes 0.8 s to transfer 100 MB of data, over the wire.

5.2.3.3 Quantitative Evaluation Results

Table 5.8 shows the evaluation results for the different scaling factors (timed and extrapolated), network bandwidths, and additionally also the overall archive sizes with the corresponding number of image DAOs. The colored cell backgrounds reflect the adherence to the 24 h constraint imposed for the maximum overall run-time of the processes, where green lies under 24 h, and red over the 24 h mark. In the metadata management benchmark in Section 5.1.2.4 we have estimated the overall archive size that can be managed with DISTARNET somewhere between 1 PB and 10 PB. In comparison, now that we also consider the bitstream data and the additional time needed for checksum calculation, network transfer, and data format migration, we see that the estimated overall archive size that we can manage on one node, lies around 10 TB. Depending on the assumed network bandwidth, either just under 10 TB with the lower assumption, or little over 10 TB with the higher bandwidth assumption.

As said earlier, the 24 h constraint is a very conservative assumption, coming from the metadata management benchmark section. For a productive running system, we can relax this assumption, if for example the DISTARNET node uses a RAID storage system, providing higher data storage reliability and thus lessening the need for frequent integrity checking. In the case with the higher network bandwidth, we could achieve archive sizes of 100 TB, if we allow the processes to run for almost a whole week, or even 1 PB if we give the node 2 months for the running of the processes.

Table 5.9 shows a more detailed representation of the evaluation results for the scaling factor $F = 1$, and the two network bandwidths. Here, we can additionally see the overall distribution of the time contributions, and also within each process mix. When looking at the overall time, then the PICP / DRP / RCPIn mix (1) is responsible for 95 % of the duration. Looking further at (1), we see that in the case of the slower network, the main time consumer is the RCPIn part of the mix (64.76 %). When looking at (1) in the case of the faster network, then the situation is reversed. The main time contribution stems from the checksum calculation including a small amount of process execution overhead. If we now look at the DFMP / RCPOut mix (2), then we will see a similar picture. In the case of the slower network, the main contribution comes from the network transfer in RCPOut with approx. 2/3, whereas the TIFF to JPEG 2000 conversion uses approx. 1/3 of the time. Here, the picture also reverses as before, when looking at the faster network scenario.

Figure 5.6 represents a different depiction for the overall time durations of the evaluation runs for both network speeds, and their distribution within each process mix.

With the overall goal to achieve a better performance of the system, a large step was done when going from a slower to a faster network connection. To further increase the performance, we need to decrease the processing time needed for checksum calculation in (1) and also of TIFF to J2P conversion in (2). One option is if we assume to be able to achieve faster processing by a factor of 10, by means of better hardware

	Scaling Factor F=1 (100 Image DAOs, 10 GB)			
	100 Mb/s		1 Gb/s	
	abs	rel	abs	rel
(1) PICP / DRP / RCPIn	**123.534 s**	**95.46 %**	**48.638 s**	**95.57 %**
(Process + Checksum)	(43.534 / 35.24 %)		(40.638 s / 83.55 %)	
(RCPIn for 1 GB)	(80 s / 64.76 %)		(8 s / 16.45)	
(2) DFMP / RCPOut	**5.881 s**	**4.54 %**	**2.256 s**	**4.43 %**
(Process Execution)	(0.111 s / 1.89 %)		(0.086 s / 4.81 %)	
(TIFF to JPEG 2000 for 100 MB)	(1.77 s / 30.10 %)		(1.77 s / 78.46%)	
(RCPOut for 50 MB)	(4 s / 68.02 %)		(0.4 s / 17.73 %)	
TOTAL	**129.415 s**	**100 %**	**50.894 s**	**100 %**

Table 5.9: Evaluation Results for $F = 1$

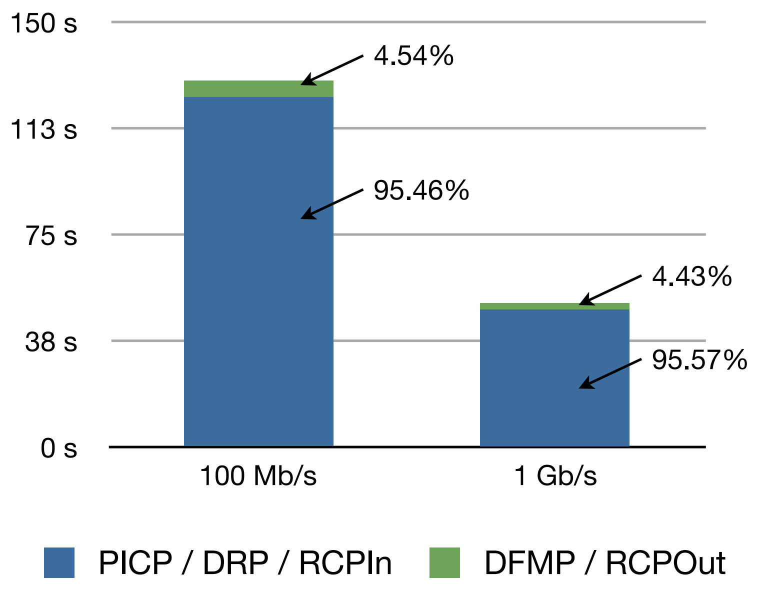

Figure 5.6: Bar Plot of the Results with Scaling Factor $F = 1$

(multiple cores, high performance disk arrays), then this would roughly cut the overall processing time by 30 % in the case of the slower network connection, and by 75 % in the case of the faster network connection. Looking at Table 5.8, this would provide a shift to the “right” for the case with the lower bandwidth, and almost a shift for the other case.

If we additionally relax the 24 h constraint in conjunction with the rise in processing power, then the 1 PB archive size can be achieved with overall processing times around 2 weeks. To shift further to the right side of the performance table, we would than again need to increase the network bandwidth. One inexpensive option to further increase “network bandwidth” between sites would be to simply ship the harddrives by courier.

Another feasible possibility to increase the overall performance of a DISTARNET site, besides better hardware, would be by deploying multiple nodes, where each node could run on “commodity hardware”, and would manage a fraction of the archive. As before the limiting factor would be the needed equivalent rise in the overall network connectivity for the whole site.

In summary, barring any measures to increase the performance of a node, when using commodity hardware with current technology, we estimate that one DISTARNET node can effectively manage around 10 TB of archived data under the 24 h constraint. Under a more relaxed constrained of 2 weeks, and using high performance hardware, we estimate that the system can manage around 1 PB of archived data.

5.3 Summary

In this chapter, we have presented and discussed three different evaluations concerning different aspects of the DISTARNET system implementation. The first, regarding triple-store usage for the management of metadata, as used by DISTARNET. Secondly, a qualitative evaluation of the implemented main DISTARNET processes, and lastly, a quantitative evaluation of the implemented DISTARNET processes.

We have defined a benchmark for process-based metadata management tailored to a long-term digital preservation setting, and we have evaluated the scalability characteristics of metadata management using the Jena triple-store to find out potential scalability restrictions. The evaluation results provided show that even with commodity hardware, archive sizes of up to approx. 5 PB can be supported with triple-store-based metadata management. This evaluation has shown that the usage of triple-stores for the storage of metadata used in a long-term preservation environment is a viable option, and will not present a limiting factor.

The second evaluation, with a focus on the qualitative aspects, has shown that the implementation of the core DISTARNET processes as described in Chapter 4 are capable of performing the tasks as they where specified and designed in Chapter 3. This result also validates the developed novel concepts, onto which the implementation is based.

The third, the quantitative evaluation of our implementation has shown what collection sizes are effectively manageable in regard to the used hardware, and time constraints for the running of the processes.

In Section 1.1 we have introduced three exemplary scenarios in which a DISTARNET system could be used. After the quantitative evaluation of the implementation of the DISTARNET system, we see that these scenarios are viable, either with some constraints regarding the collection size, or with added hardware and network requirements in conjunction with a relaxed maximal process running time.

In the scenarios of a Multinational Pharmaceutical Corporation, and the National Museum of History & Native Art, the main constraint will be the requirement for high-speed network connectivity between the DISTARNET nodes. While this requirement also imposes a constraint in the scenario with the Cloud Storage Provider, when "only" deployed on a per data-center basis, the added network speed requirement could be met, allowing large collection sizes. If we additionally relax the maximum process running time, then all three scenarios can provide a large collection sizes in the region of 1 PB.

6 Related Work

In this chapter, we discuss the similarities and differences between DISTARNET and the representatives of three categories of systems. The list of projects in each category is not exhaustive and should only provide an overview. The first category, *Digital Library*, contains approaches for providing digital library services in a distributed environment consisting of cooperating nodes. The second category, *Digital Repository - Archive*, contains projects which are closing the gab between digital repositories as managers of the holdings, and the archiving services providing long-term preservation. The third category, the *Digital Archive*, contains projects providing management of the holdings fused together with archiving services, providing long-term preservation.

Digital Library

In this category, we will mention two representatives, BRICKS [97] and DILIGENT [98] / D4Science [99], which are projects for providing library services in a distributed environment, in which the participating nodes build a cooperating network. BRICKS is a P2P-based digital library that provides, management, easy access, and sharing of content and resources between participants. DILIGENT/D4Science is a Grid-based digital library infrastructure providing a virtual research environment, in which users can manage, share content and resources, and collaborate with each other.

Both projects allow flexible organization of materials with support for complex digital objects, user annotations, and building of collections/subcollections. These features are also found in DISTARNET. Also, similar is the distributed cooperative nature which these projects share with DISTARNET. The main difference in regard to DISTARNET is that these projects, and digital libraries in general, do not provide long-term preservation services.

Digital Repository - Archive

Fedora Commons [100] and DSpace [101] are open source projects that allow the creation of digital repositories for management of digital objects. Both provide support for complex objects, where Fedora Commons additionally has a more flexible data model, which is partly based on RDF. DSpace and Fedora Commons allow only simple management of the archived materials where collections/subcollections can be created. More advanced management features like creating annotations, as supported by the DISTARNET data model, is not supported by both projects. As Fedora Commons is a framework which is meant to be used to create individual digital repository solutions, there are projects working on creating an annotations framework for Fedora Commons which integrates the Open Annotations Data Model[1].

To close the gap between the management of digital holdings and preservation services, these two projects have merged together into DuraSpace [102], which is creating DuraCloud [103]. DuraCloud provides both storage and access services, including content replication and monitoring services that can span multiple cloud-storage providers. The idea is that digital repositories (e.g., Fedora, DSpace) using DuraCloud can expand their systems, and provide long-term preservation capabilities, or individuals can use DuraCloud directly through the DuraCloud UI, as a long-term preservation system. DuraCloud and DISTARNET are similar as both provide processes for the management of digital objects necessary for digital long-term preservation. The main difference lies in the location of the preservation services. In the case of DuraCloud they are completely situated in the cloud, and allow no cooperation with other institutions while in the case of DISTARNET they are situated locally on the premises of each cooperating site in the network.

SHERPA DP (Securing a Hybrid Environment for Research Preservation and Access, Digital Preservation) [104] describes a preservation framework which distinguishes between institutional repositories (e.g., DSpace, Fedora) acting as content providers, and between external service providers with expert knowledge in long-term preservation. The service providers would provide long-term archiving ser-

[1]`http://www.openannotation.org`

vices, for the content providers, which would transfer the data and metadata into the preservation archive of the service provider right after it is ingested into the content providers repository. In the case of the SHERPA DP architecture, the preservation services are defined to be build centrally, and shared between different institutional repositories.

Both of the presented project can be seen as use case for DISTARNET, where the institutional repositories could cooperatively run a preservation network, which would then serve the preservation services to the individual institutional repositories.

Digital Archive

LOCKSS (Lots Of Copies Keep Stuff Safe) [105] provides digital preservation through high replication in a peer-to-peer network. The main focus lies on the preservation of journals. The idea behind LOCKSS is to provide long-term preservation through a larger number of cooperating nodes, where for each object at least 7 copies are kept in the network, needed to provide recovery and repair in case damage occurs. The LOCKSS software can be used to build open or closed networks. This project shares the common idea with DISTARNET for a cooperating network, and using shared resources to provide preservation services. The followed cooperative and distributed approach allows generally lower infrastructure, and maintenance costs, the ability for virtually unlimited growth, and also a higher degree of availability and reliability, as the archiving infrastructure is distributed among the participants, and the overall resources used are shared among all participants. However, the focus of the LOCKSS project is on journals, and it also does not provide advanced access functionalities like annotations, collection/subcollection management, or data format migration functionalities, like in the case of DISTARNET.

Cheshire 3 [106] focuses on building a component based framework integrating data grid (e.g., Storage Resource Broker [107]), digital repositories (e.g., DSpace and Fedora), and text and data mining systems (e.g., Cheshire). The similarities to DISTARNET are the addressing of the twin aspects of access, e.g., support for complex objects, collection/subcollection management, etc., and long-term preservation services, by employing distributed technology for large-scale support.

kopal (Co-operative Development of a Long-Term Digital Information Archive) [108] is a long-term archiving solution for electronic publications. It consists of koLibRI (kopal Library for Retrieval and Ingest) and DIAS-Core, which is the core of IBM's

Digital Information Archiving System developed together with the National Library of the Netherlands (Koninklijke Bibliotheek [KB]) as e-Depot [109]. It provides similar functionality to DISTARNET by including support for complex objects, collection/-subcollection management, etc., and long-term preservation services like replication, integrity checking, and data format migration. Additionally, it provides support for integrating emulation services. The main difference lies in its architecture as it is built as a centralized system, not allowing to collaboratively share resources with other institutions.

CASPAR (Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval) [110] is a long-term preservation service-based framework, with a strong emphasis on preservation of representation information, the preservability of the infrastructure itself, and validation of the proposed concepts. The strong focus on preservation of representation metadata goes beyond of the features of DISTARNET. While DISTARNET provides a data model which supports the preservation of representation information with each object, the CASPAR approach additionally defines a whole infrastructure for creating, registering, storing, and preserving extensive representation metadata.

PLANETS (Preservation and Long-term Access through Distributed NETworkS) [111], built on a service oriented architecture, provides a suite of software tools and services to support the preservation and long-term access to digital content. The main components are Plato, the Testbed, and the Interoperability Framework. Plato [112, 113] provides preservation planning services, which interactively help the user to identify the characteristics of the digital objects, and the best course of action. The Testbed contains a corpus of 5000 sample digital objects, which can then be used to test workflows or tools beforehand. The interoperability framework provides standard services such as authorization, authentication, orchestration, data and metadata management. Additionally, through its service-based architecture allows the integration of third-party services (e.g., emulation services). Similarly to DISTARNET, the PLANETS approach also supports complex objects, advanced digital object management functionalities, and preservation processes providing integrity checking, and data format migration.

SHAMAN (Sustaining Heritage Access through Multivalent ArchivingNg) [114], uses iRODS [115] a integrated rule-oriented data grid (successor to SRB) as the implementation technology, which provides transparent support for local and remote storage, i.e., allowing to additionally leverage the Cloud [116] and use it for storage. It provides a distributed, preservation policy, and workflow driven preservation environment, with a strong focus on preservation of context, discoverability of the holdings, and risk management through geographically dispersed replication support. Like DISTARNET, this project provides support for complex objects, advanced digital object management functionalities, and long-term preservation processes, providing integrity checking, replication, and data format migration. The main difference lies in its larger orientation as a framework, from which individual solutions can be developed. For example, it would be theoretically possible to implement a preservation solution with SHAMAN, in which the DISTARNET concepts and processes are implemented as workflows.

Hoppla [117] is an archiving solution that combines back-up and fully automated migration services for data collections in environments with limited expertise, and resources for digital preservation. By employing data format identification (e.g., DROID) and characterization tools (e.g., JHOVE), and through externally defined rules, the system provides file format migration services that will ensure long-term readability and usability of the digital objects. The project focuses on automated data format migration with expert knowledge provided as external services, whereas DISTARNET provides data format migration capabilities, but requires an expert user. Also, Hoppla depends on the locally provided storage, and would need to be combined with some other service with replication and integrity checking features, for providing long-term preservation such as DISTARNET does.

7 Conclusion and Future Work

In the following chapter, we conclude the work, and provide an overview over a number of topics which we deem interesting to be followed on in the future.

7.1 Conclusion

By combining P2P and Grid technology, the developed DISTARNET concepts describe a fully distributed, fault tolerant archiving environment with process-based autonomic behavior governed by preservation policies. Through self-configuration, self-healing, and self-optimization capabilities, the environment autonomously provides dynamic replication, automated consistency checking, and recovery of the archived digital objects. By developing a highly flexible data model and the specification of sophisticated management processes, the environment provides support for complex data objects, user generated annotations, collections, and arbitrary links.

By deploying a DISTARNET System, an implementation based on these concepts, collaborating geographically dispersed organizations can build an Internet-based long-term preservation system that meets the requirements for reliable and fault-tolerant management of large digital archives.

Replication and Distribution

DISTARNET provides a fully distributed environment, where collaborative geographically dispersed organizations can share resources, and through automated processes provide secure and reliable replication and distribution of the archive materials.

Fault Class	Failure	Detection	Reaction
Distributed	Node Loss,	PNCP	ADRP
Infrastructure	Node Dependability	PNCP	ADRP
Content	DAO Corruption	PICP	DRP
	Data Format obsolescence	PICP	DFMP
Node Engine	Process Execution Failure	PEL	execution of corresponding recovery process

Table 7.1: DISTARNET Failure Recovery

Fault-Tolerance and Failure Management

Due to the continuous monitoring of nodes, the DISTARNET system detects abnormal conditions or problems that may harm its proper functioning and is able to recover automatically from those situations, by means of predefined processes. Table 7.1 shows a classification of faults from which a DISTARNET can recover.

Hardware Problems caused by failure (e.g., power failure, hardware failure, etc) or disaster (e.g., natural disaster, fire) can result in the destruction of the whole node, or in the destruction or corruption of the stored information object containers. The loss of a node is discovered through the *PNCP* which triggers a *Node-Lost Event* after some predefined period of time because at first a *Network Problem* is assumed. In the case of a *Node-Lost Event* this information is stored in the *Node Information Repository*, the *Replica Location Repository* is updated, and both are propagated throughout the network. Subsequently, the ADRP is started for the DAOs that are affected (network wide redundancy) by the lost node.

Network Problems caused by lost network connection or an intermittent network connection are detected through the *PNCP*. For a lost connection to a node to be classified as a network problem, upon the return of a node it has to be verified that the node was running, only the network was down, and all DAOs are accounted for. Such discovered network problems trigger an entry into the Node Information Repository and are taken into consideration (e.g., reliability of a node) by the ADRP.

Problems with the *Content* of the archive caused through the corruption (e.g., hardware problems, malicious acts, etc.) of the DAOs is discovered through the *PICP*. Any detected corruption is logged correspondingly in the *Node Information Repository* where it speaks about the reliability of a specific node, in the *Replica Location Repository*, and in the audit trail of the DAO for future reference. Also, subsequently the DRP is triggered to resolve the problem.

DISTARNET processes are able to *Self Recover* from problems encountered during their execution. The PEL responsible for the execution of the processes, monitors and logs every step of the execution. In the event of failure of any step, the corresponding recovery action is triggered.

Management of Complex Information Objects

DISTARNET supports *Complex Information Objects* by providing a flexible data model, and maintenance processes that can manage complex object. The *Integrity* of each DAO is guaranteed through periodic integrity checking, and automated repair. The *Interpretability* of the logical representation of the DAOs is guaranteed by processes providing automated data format migration and is supported by the data model.

Scalability

Through the distributed nature of the system and the maintenance processes, DISTARNET provides vertical scalability, i.e., size of the archived collection, and horizontal scalability, i.e., number of collaboration sites. The quantitative evaluation results show that even with commodity hardware, archive sizes of up to approx. 1 PB can be achieved.

Openness and Extensibility

Openness and extensibility are provided though the modular nature of the system, where future extension or exchange of the modules is supported. Additionally, the flexible data model can adapt to new needs that can arise in the future (e.g., in case of new data formats, new metadata standards, etc.).

Resource Discovery and Load Balancing

Resource discovery and load balancing are supported through the fully distributed design of the network, adaptability of the triggers to lower the overall load on the nodes caused by the maintenance processes, and by the ADRP which optimizes the usage of the storage resources provided in the network.

Authentication, Authorization, and Auditing

Authentication and authorization are supported by the virtual organization-based organization of the DISTARNET Sub-Networks in which each participant needs to be authenticated and authorized to be part of the network. Auditing is supported by the data model, which allows the storage of an audit trail within every DAO, and thus receiving the same long-term care as the DAO itself.

7.2 Future Work

In the following section, we first discuss a few topics which are more technical in nature, and pertain to the current implementation. Afterwards, we provide an overview over a number of topics going beyond our current work, which we find interesting and important that possibly warrant future research efforts in those directions.

Extending the Current Implementation

User-Facing Functionalities

The conceptual development and implementation effort was limited to the development of core functionalities representing our vision of a distributed archival network. As such, the implementation does not provide an user interface, and the corresponding modules need to be extended to provide a visual interface providing ingest, management, and access functionality to the user.

System Functionalities

To optimize the Data Format Migration Process (DFMP), remote execution needs to be addressed as it would be more efficient to perform DFM on remote nodes where the data is stored. Further, how can it be ensured, that the remote node has the capabilities to execute the migration scripts, and how can the remote node be protected from possible maliciously behaving scripts? Could some form of sandboxing be used to execute the scripts in a controlled environment?

As we have multiple copies of a DAO distributed around the network, how can we provide merging or synchronizing of DAOs if changed on two or more nodes? Further, how can we provide versioning for any changes applied to a DAO?

Future Topics

Semantic Digital Archives, Context, and Process Preservation

The current development in the digital library community is going toward Semantic Digital Libraries [118]. This development should also be followed in the digital preservation community leading to research in the direction of Semantic Digital Archives (SDA). We understand SDAs as a further development of current ideas for a Digital Preservation System, combined with Semantic Web technologies, with the goal to allow not only access to the archive on the basis of archival metadata, but also according to the connections the archived objects have with internally or externally defined concepts. The Data Model implementation of DISTARNET is based on RDF, providing a good basis for such further development.

In digital preservation, context is identified as a critical aspect of preservation metadata. An extensive discussion outlining the importance of context in digital preservation is provided in [119]. Under context, we understand all metadata which describe the objects origins, composition, and purpose. A development toward SDAs can also facilitate the capturing of the context for the archived digital objects. Further, as the designated communities evolve, their knowledge expands, so will also the usage of the digital objects, and thus the context evolve over time. Means need to be developed to also allow to capture this change over time.

The amalgamation of Web 3.0 and long-term digital preservation is a big step, but we need to go further. At present, we preserve static objects, or in the best case, a static representation of something dynamic. Following [120], the future development needs to proceed toward the preservation of processes. At present, for example, when some experiment, or some other arbitrary process is executed, we are presented with results in the form of data, which are then preserved together with metadata describing the context. The goal now would be to preserve the whole execution chain of the process for the future, allowing assessment of their authenticity upon re-execution, and capturing the whole context.

Ontology-Based Process Definitions

The defined and executed preservation processes represent an important part of every preservation environment. In the DISTARNET project, we developed them conceptually and afterwards realized them in the implementation. Even if we use open-source software libraries, eventually, the implementation will become obsolete, and will need to be exchanged. Also, the implementation we provide is highly specific to our implementation and can not be shared without a significant effort with other projects.

The goal now would be to provide a conceptual description of the DISTARNET processes as an ontology following the work in [121]. An ontology provides the needed expressive power, and facilitates information reuse. It could be further integrated with other internal ontologies, shared and extended in order to incorporate future changes. The use of established standards in the development of the ontology, such as the OWL language, will ensure that the defined processes defy obsoleteness in the future. The instances of the process definitions could then be retrieved using SPARQL queries and executed by a custom Process Execution Logic.

Further, the now conceptually defined processes should also be validated, to guarantee their executability. Also, as DISTARNET performs preservation actions automatically, a form of validation of the executed actions against the preservation plan stored in the ontology, would provide additional feedback if the system is behaving as it should.

Bibliography

[1] "Reference Model for an Open Archival Information System (OAIS) - Blue Book," Consultative Committee for Space Data Systems (CCSDS), Tech. Rep. January, 2002. [Online]. Available: http://nssdc.gsfc.nasa.gov/nost/isoas/us19/650x0_20010226rl.pdf 1, 2.2

[2] "Reference Model for an Open Archival Information System (OAIS) - Pink Book - Draft," Consultative Committee for Space Data Systems (CCSDS), Tech. Rep. August, 2009. [Online]. Available: http://public.ccsds.org/sites/cwe/rids/Lists/CCSDS6500P11/Attachments/650x0p11.pdf 1, 2.2

[3] R. Moore, "Towards a theory of digital preservation," *International Journal of Digital Curation*, vol. 3, no. 1, pp. 63–75, 2008. [Online]. Available: http://ijdc.net/index.php/ijdc/article/view/63 2.1.1

[4] M. Mois, C.-p. Klas, and M. H. F.-h. De, "DIGITAL PRESERVATION AS COMMUNICATION WITH THE FUTURE Faculty for Mathematics and Computer Science FernUniversität in Hagen," *Science*, 2009. 2.1.1

[5] T. Kuny, "A Digital Dark Ages ? Challenges in the Preservation of Electronic Information," in *63rd IFLA General Conference*, 1997. 2.1.1

[6] J. Rothenberg, "Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation. A Report to the Council on Library and Information Resources," Tech. Rep., 1999. [Online]. Available: http://eric.ed.gov/ERICWebPortal/recordDetail?accno=ED426715 2.1.1, 2.1.3

[7] D. Swade, "The Problems of Software Conservation," *Resurrection, Computer Conservation Society*, no. 7, 1993. [Online]. Available: http://www.cs.man.ac.uk/CCS/res/res07.htm#f 2.1.3

[8] "Roseta Project." [Online]. Available: http://rosettaproject.org 2.1.3

[9] K. Wendel and W. Schwitin, "Eine Arche zur Rettung digitalen Kulturguts," Tech. Rep. September 2006, 2007. [Online]. Available: http://elib.uni-stuttgart.de/opus/volltexte/2007/3011/pdf/ARCHE_v1.6_25.01.2007.pdf 2.1.3

[10] A. Amir, F. Müller, and P. Fornaro, "Towards a channel model for microfilm," *Proceedings of IS&T's Archiving Conference*, no. 1, 2008. [Online]. Available: https://www.math.uzh.ch/fileadmin/user/aireeal/publikation/peviar_microfilm.pdf 2.1.3

[11] F. Müller, P. Fornaro, L. Rosenthaler, and R. Gschwind, "PEVIAR: Digital Originals," *Journal on Computing and Cultural Heritage*, vol. 3, no. 1, pp. 1–12, Jun. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1805961.1805963 2.1.3

[12] S. Schilke and A. Rauber, "Long-term archiving of digital data on microfilm," *International Journal of Electronic Governance*, vol. 3, no. 3, 2010. [Online]. Available: http://inderscience.metapress.com/index/T133878207N8873W.pdf 2.1.3

[13] R. A. Lorie, "Long term preservation of digital information," in *Proceedings of the first ACM/IEEE-CS joint conference on Digital libraries - JCDL '01*. New York, New York, USA: ACM Press, 2001, pp. 346–352. [Online]. Available: http://portal.acm.org/citation.cfm?doid=379437.379726 2.1.3

[14] "The Computer History Simulation Project." [Online]. Available: http://simh.trailing-edge.com 2.1.3

[15] E. Brewer, C. Dellarocas, A. Colbrook, and W. Weihl, "Proteus: A high-performance parallel-architecture simulator," Tech. Rep., 1991. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.7487 2.1.3

[16] M. Rosenblum, E. Bugnion, S. Devine, and S. a. Herrod, "Using the SimOS machine simulator to study complex computer systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 1, pp. 78–103, Jan. 1997. [Online]. Available: http://portal.acm.org/citation.cfm?doid=244804.244807 2.1.3

[17] S. Herrod, "Using complete machine simulation to understand computer system behavior," Ph.D. dissertation, 1998. [Online]. Available: ftp://reports.stanford.edu/www/pub/public_html/cstr/reports/cs/tr/98/1603/CS-TR-98-1603.pdf 2.1.3

[18] M. Burnet and R. Supnik, "Preserving Computing's Past : Restoration and Simulation," *Digital Technical Journal*, vol. 8, no. 3, pp. 23–38, 1996. [Online]. Available: http://ftp.se.scene.org/pub/bitsavers.org/simh.trailing-edge.com/docs/dtjn02pf.pdf 2.1.3

[19] J. Rothenberg, "An Experiment in Using Emulation to preserve Digital Publications," Koninklijke Bibliotheek, Tech. Rep., 2000. [Online]. Available: http://www.studioautomata.com/itp/indestudy/emulationpreservationreport.pdf 2.1.3

[20] R. A. Lorie, "Long-Term Archiving of Digital Information," IBM, Tech. Rep. May, 2000. [Online]. Available: http://www.imaginar.org/dppd/DPPD/138ppLongterm.pdf 2.1.3

[21] J. van der Hoeven, B. Lohman, and R. Verdegem, "Emulation for Digital Preservation in Practice: The Results," *International Journal of Digital Curation*, vol. 2, no. 2, 2007. [Online]. Available: http://www.ijdc.net/index.php/ijdc/article/view/50 2.1.3

[22] M. Guttenbrunner, C. Becker, A. Rauber, and C. Kehrberg, "Evaluating strategies for the preservation of console video games," in *Proceedings of The Fifth International Conference on Preservation of Digital Objects*, 2008, pp. 115–121. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.1417&rep=rep1&type=pdf 2.1.3

[23] M. Guttenbrunner and A. Rauber, "Design Decisions in Emulator Construction: A Case Study on Home Computer Software Preservation," in *8th International Conferece on Preservation of Digital Objects*, 2011, pp. 171–180. [Online]. Available: http://publik.tuwien.ac.at/files/PubDat_201169.pdf 2.1.3

[24] Library of Congress, "METS - Metadata Encoding and Transmission Standard." [Online]. Available: http://www.loc.gov/standards/mets/ 2.3.1

[25] M. V. Cundiff, "An introduction to the Metadata Encoding and Transmission Standard (METS)," *Library Hi Tech*, vol. 22, no. 1, pp. 52–64, 2004. [Online]. Available: http://www.emeraldinsight.com/10.1108/07378830410524495 2.3.1

[26] L. Cantara, "METS : The Metadata Encoding and Transmission Standard," *Cataloging & Classification Quarterly*, vol. 40, no. 3-4, pp. 237–258, 2009. [Online]. Available: http://dx.doi.org/10.1300/J104v40n03_11 2.3.1

[27] R. Guenther, R. Wolfe, O. Brandt, M. Enders, T. Habing, F. Lazzarino, C. Redding, and J. Riley, "Guidelines for using PREMIS with METS for exchange," Library of Congress, Tech. Rep., 2008. 2.3.1

[28] P. Caplan, "Understanding PREMIS," 2009. [Online]. Available: http://www.loc.gov/standards/premis/understanding-premis.pdf 2.3.1

[29] NISO, "Data Dictionary - Technical Metadata for Digital Still Images," Tech. Rep., 2006. 2.3.1

[30] Library of Congress, "textMD." [Online]. Available: http://www.loc.gov/standards/textMD/ 2.3.1

[31] T. Steinke, "LMER, Long-term preservation Metadata for Electronic Resources," Die Deutsche Bibliothek, Tech. Rep. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:1111-2005051906 2.3.1

[32] S. Sugimoto, T. Baker, M. Nagamori, T. Sakaguchi, and K. Tabata, "Versioning the Dublin Core across multiple languages and over time," *Proceedings 2001 Symposium on Applications and the Internet Workshops (Cat. No.01PR0945)*, pp. 151–156, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=998223 2.3.1

[33] D. Hillmann, "Using Dublin Core," 2005. [Online]. Available: http://dublincore.org/documents/2005/11/07/usageguide/ 2.3.1

[34] E. Mylonas and A. Renear, "The text encoding initiative at 10: not just an interchange format anymore - but a new research community," *Computers and the Humanities*, pp. 1–9, 1999. [Online]. Available: http://www.springerlink.com/index/Q70370679U1M7027.pdf 2.3.1

[35] JSTOR and Harvard University Library, "JHove - JSTOR/Harvard Object Validation Environment." [Online]. Available: http://jhove.sourceforge.net 2.3.2

[36] National Archives of UK, "DROID / PRONOM." [Online]. Available: http://www.nationalarchives.gov.uk/aboutapps/pronom/ 2.3.2

[37] OAI, "Open Archives Initiative." [Online]. Available: http://www.openarchives.org 2.3.3

[38] J. Bekaert and H. V. de Sompel, "Access interfaces for open archival information systems based on the OAI-PMH and the OpenURL framework for context-sensitive services," in *Ensuring Long-term Preservation and Adding Value to Scientific and Technical data (PV 2005)*, 2005. [Online]. Available: http://www.ukoln.ac.uk/events/pv-2005/pv-2005-final-papers/032.pdf 2.3.3

[39] C. Lagoze, H. Van de Sompel, M. Nelson, and S. Warner, "The Open Archives Initiative Protocol for Metadata Harvesting," 2008. [Online]. Available: http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm 2.3.3

[40] C. Lagoze, H. van de Sompel, P. Johnston, M. Nelson, R. Sanderson, and S. Warner, "ORE User Guide - Primer," Open Archives Initiative, Tech. Rep., 2008. [Online]. Available: http://www.openarchives.org/ore/ 2.3.3

[41] A. Oram, *Peer-to-peer: Harnessing the power of disruptive technologies*, 2001, vol. 32, no. 2. 2.4.1

[42] D. Schoder, K. Fischbach, and R. Teichmann, *Peer-to-peer. Oekonomische, technologische und juristische Perspektiven*. Springer Verlag, Heiderlberg, Germany, 2002. 2.4.1

[43] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999. 2.4.2

[44] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, pp. 200–222, Aug. 2001. [Online]. Available: http://hpc.sagepub.com/cgi/doi/10.1177/109434200101500302 2.4.2

[45] Gartner, "Service Oriented' Architectures, Part 1, SSA Research Note SPA-401-068," Tech. Rep., 1996. [Online]. Available: http://www.gartner.com/id=302868 2.4.3

[46] ——, "Service Oriented' Architectures, Part 2, SSA Research Note SPA-401-069," Tech. Rep., 1996. [Online]. Available: http://www.gartner.com/id=302869 2.4.3

[47] OASIS, "OASIS Reference Model for Service Oriented Architecture 1.0," 1996. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf 2.4.3

[48] P. Bianco, R. Kotermanski, and P. Merson, "Evaluating a service-oriented architecture," no. September, 2007. [Online]. Available: http://repository.cmu.edu/sei/324/ 2.4.3

[49] P. Mell and T. Grance, "The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standarts and Technology," *National Institute of Standards and Technology US . . .*, 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf 2.4.4

[50] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," *ISW-2000*, 2000. [Online]. Available: http://www.cert.org/research/isw/isw2000/papers/56.pdf 2.5

[51] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335465 2.5, 2.5.2

[52] B. Randell, P. Lee, and P. C. Treleaven, "Reliability Issues in Computing System Design," *ACM Computing Surveys*, vol. 10, no. 2, pp. 123–165, 1978. [Online]. Available: http://portal.acm.org/citation.cfm?doid=356725.356729 2.5

[53] B. Selic, "Fault Tolerance Techniques for Distributed Systems," 2006. 2.5.1

[54] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985. [Online]. Available: http://wrap.warwick.ac.uk/26189/ 2.5.2

[55] P. Jalote, *Fault Tolerance in Distributed Systems*. Prentice Hall, 1994. 2.5.2

[56] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern-Oriented Software Architecture: A System of Patterns," 1996. 2.5.3

[57] A. Avizienis, "The N-version approach to fault-tolerant software," *Software Engineering, IEEE Transactions on*, no. 12, pp. 1491–1501, 1985. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1701972 2.5.4, 2.5.4

[58] J. J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell, "A Program Structure for Error Detection and Recovery," in *Lecture Notes in Computer Science*, vol. 16. Springer-Verlag, 1974, pp. 171–187. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.9163&rep=rep1&type=pdf 2.5.4, 2.5.4

[59] J. C. Laprie, J. Arlat, C. Béounes, K. Kanoun, and C. Hourtolle, "Hardware and Software Fault Tolerance: Definition and analysis of architectural solutions," in *Proc l7th Int Symp on FaultTolerant Computing FTCS17*. IEEE Computer Society Press, 1987, pp. ll6–121. 2.5.4, 2.5.4

[60] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988. [Online]. Available: http://portal.acm.org/citation.cfm?id=47054.47058 2.5.4, 2.5.4

[61] E. N. Adams, "Optimizing preventive service of software products," *IBM Journal of Research and Development*, vol. 28, no. 1, pp. 2–14, 1984. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5390362 2.5.4, 2.5.4

[62] I. Lee and R. K. Iyer, "Software dependability in the Tandem GUARDIAN system," *Software Engineering IEEE Transactions on*, vol. 21, no. 5, pp. 455–467, 1995. 2.5.4, 2.5.4

[63] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: analysis, module and applications," in *TwentyFifth International Symposium on FaultTolerant Computing Digest of Papers*, vol. 4. IEEE Comput. Soc. Press, 1995, pp. 381–390. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=466961 2.5.4, 2.5.4

[64] K. M. Chandy and C. V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs," *IEEE Transactions on Computers*, vol. 21, no. 6, pp. 546–556, 1972. [Online]. Available: http://dl.acm.org/citation.cfm?id=1638603.1638989 2.5.4, 2.5.4

[65] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, "Minimizing completion time of a program by checkpointing and rejuvenation," in *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems SIGMETRICS 96*, vol. 24, no. 1. ACM Press, 1996, pp. 252–261. [Online]. Available: http://portal.acm.org/citation.cfm?doid=233013.233050 2.5.4, 2.5.4

[66] P. Jalote, Y. Huang, and C. Kintala, "A Framework for Understanding and Handling Transient Software Failures," in *2nd ISSAT Intl. Conf. on Reliability and Quality in Design*, Orlando, Florida, 1995. 2.5.4

[67] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. January, pp. 41–50, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1160055 2.6, 2.6.1, 2.6.2, 2.6.3

[68] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001. [Online]. Available: http://www.nature.com/doifinder/10.1038/scientificamerican0501-34 2.7

[69] W3C, "W3C Semantic Web." [Online]. Available: http://www.w3.org/2001/sw/ 2.7

[70] T. Berners-Lee, "Universal Resource Identifiers in WWW," IETF, Tech. Rep., 1994. [Online]. Available: http://tools.ietf.org/html/rfc1630 2.7.1

[71] ——, "Uniform Resource Identifier (URI): Generic Syntax," IETF, Tech. Rep., 2005. [Online]. Available: http://tools.ietf.org/html/rfc3986 2.7.1

[72] F. Manola and E. Miller, "RDF Primer," Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/rdf-primer/ 2.7.2

[73] E. Miller, "An introduction to the resource description framework," *Bulletin of the American Society for Information Science*, pp. 15–19, 2005. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/bult.105/full 2.7.2

[74] D. Beckett and T. Berners-Lee, "Turtle - Terse RDF Triple Language," Tech. Rep., 2012. [Online]. Available: http://www.w3.org/TR/turtle/ 2.7.2

[75] J. Grant and D. Beckett, "RDF Test Cases," W3C, Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/rdf-testcases/ 2.7.2

[76] T. Berners-Lee, "Primer: Getting into RDF & Semantic Web using N3," Tech. Rep., 2000. [Online]. Available: http://www.w3.org/2000/10/swap/Primer.html 2.7.2

[77] D. Brickley and R. von Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/rdf-schema/ 2.7.3

[78] W3C, "SPARQL Query Language," 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/ 2.7.4

[79] W3C OWL Working Group, "OWL 2 Web Ontology Language Document Overview (Second Edition)," Tech. Rep., 2012. [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-overview-20121211/ 2.7.5

[80] I. Subotic, S. Margulies, and L. Rosenthaler, "DISTARNET: Distributed Archiving Network," in *Proceedings of Archiving 2006*, vol. 3, 2006, pp. 131–134. 3, 4.5

[81] CRL, OCLC, and NARA, "Trustworthy Repositories Audit & Certification: Criteria and Checklist," 2007. [Online]. Available: http://wiki.digitalrepositoryauditandcertification.org/pub/Main/ReferenceInputDocuments/trac.pdf 3.1

[82] H. Schuldt, G. Alonso, C. Beeri, and H. Schek, "Atomicity and isolation for transactional processes," *ACM Transactions on . . .*, vol. 27, no. 1, pp.

63–116, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=507236 3.2.3, 3.5.9

[83] A. Hinze, K. Sachs, and A. Buchmann, "Event-based applications and enabling technologies," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*. Nashville, Tennessee: ACM New York, NY, USA, 2009, pp. 1–15. [Online]. Available: http://doi.acm.org/10.1145/1619258.1619260http://portal.acm.org/citation.cfm?doid=1619258.1619260 3.4.4.6

[84] L. Candela, D. Castelli, N. Ferro, Y. Ioannidis, G. Koutrika, C. Meghini, P. Pagano, S. Ross, D. Soergel, M. Agosti, and Others, "The DELOS Digital Library Reference Model. Foundations for Digital Libraries," 2007. [Online]. Available: http://www.delos.info/files/pdf/ReferenceModel/DELOS_DLReferenceModel_0.98.pdf 3.6

[85] L. Voicu and H. Schuldt, "The Re : GRIDiT Protocol : Correctness of Distributed Concurrency Control in the Data Grid in the Presence of Replication," 2008. 3.6.1

[86] C. Hewitt, P. Bishop, and R. Steiger, "A Universal Modular Actor Formalism for Artificial Intelligence," in *International joint conference on Artificial intelligence*, 1973, pp. 235–245. [Online]. Available: http://dl.acm.org/citation.cfm?id=1624804 4.2.1

[87] M. Odersky, P. Altherr, V. Cremet, I. Dragos, and G. Dubochet, "An Overview of the Scala Programming Language," EPFL Lausanne, Switzerland, Tech. Rep., 2004. [Online]. Available: http://www.scala-lang.org/docu/files/ScalaOverview.pdf 4.2.2

[88] C. Sayers and A. H. Karp, "Computing the digest of an RDF graph," Tech. Rep., 2004. [Online]. Available: https://www.hpl.hp.com/techreports/2003/HPL-2003-235R1.pdf 4.5.3

[89] C. Bizer and A. Schultz, "The berlin sparql benchmark," *Int J Semantic Web Inf Syst*, 2009. [Online]. Available: http://citeseerx.ist.psu.

edu/viewdoc/download?doi=10.1.1.161.8030&rep=rep1&type=pdfpapers2://publication/uuid/9A017750-3A0A-4540-9E7A-B81F6C6330EF 5.1

[90] D. Abadi, A. Marcus, S. Madden, and K. Hollenbach, "Using the Barton libraries dataset as an RDF benchmark," Citeseer, Tech. Rep., 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.9927&amp;rep=rep1&amp;type=pdf 5.1

[91] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, Oct. 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1570826805000132 5.1

[92] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SP2Bench: A SPARQL Performance Benchmark," *CoRR*, vol. abs/0806.4, 2008. [Online]. Available: http://arxiv.org/pdf/0806.4627 5.1

[93] L. Rosenthaler, "Virtual Research Environments - A new approach for dealing with digitized sources in research in Arts and Humanities," *Des manuscrits antiques à l'ère digitale. Lectures et littératies, 23.-25. August 2011, University Lausanne.* 5.1.1.1

[94] A. Carusi and T. Reimer, "Virtual Research Environment Collaborative Landscape Study," JISC, Tech. Rep. January, 2010. [Online]. Available: http://www.jisc.ac.uk/media/documents/publications/vrelandscapereport.pdf 5.1.1.1

[95] B. Schroeder, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you," *Proceedings of the 5th USENIX Conference*, 2007. [Online]. Available: http://www.usenix.org/event/fast07/tech/schroeder/schroeder.pdf 5.1.1.3

[96] E. Pinheiro and W. Weber, "Failure trends in a large disk drive population," in *Proceedings of FAST 2007*, no. February, 2007. [Online]. Available: http://static.usenix.org/event/fast07/tech/full_papers/pinheiro/pinheiro_html/ 5.1.1.3

[97] B. Haslhofer and P. Knezevic, "The BRICKS Digital Library Infrastructure," in *Semantic Digital Libraries*, S. R. Kruk and B. McDaniel, Eds., no. Ist 507457, 2009, pp. 151–161. 6

[98] D. Castelli, L. Candela, P. Pagano, and M. Simi, “DILIGENT: A Digital Library Infrastructure for Supporting Joint Research,” in *2005 IEEE International Symposium on Mass Storage Systems and Technology*, no. Section 2. IEEE, 2005, pp. 56–59. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1612465 6

[99] D. Castelli and J. Michel, “D4Science - Deploying Virtual Research Environments,” *ERCIM News*, no. 74, pp. 8–9, 2008. 6

[100] “FedoraCommons.” [Online]. Available: http://www.fedora-commons.org 6

[101] R. Tansley and M. Bass, “DSpace as an open archival information system: Current status and future directions,” *Research and advanced technology for*, no. November 2002, pp. 446–460, 2003. [Online]. Available: http://www.springerlink.com/index/HDEPD4443HL00K4K.pdf 6

[102] “DuraSpace.” [Online]. Available: http://duraspace.org 6

[103] “DuraCloud.” [Online]. Available: http://www.duracloud.org 6

[104] G. Knight, “SHERPA DP: establishing an OAIS-compliant preservation environment for institutional repositories,” *Digital repositories: interoperability and common*, pp. 26 – 29, 2005. [Online]. Available: http://delos-wp5.ukoln.ac.uk/forums/dig-rep-workshop/knight.pdf 6

[105] V. Reich and D. Rosenthal, “Lockss (lots of copies keep stuff safe),” *New Review of Academic Librarianship*, vol. 6, pp. 155–161, 2000. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/13614530009516806 6

[106] P. Watry, “Cheshire 3 framework white paper: implementing support for digital repositories in a data grid environment,” in *International Symposium on Mass Storage Systems and Technology*, 2005, pp. 60–64. 6

[107] C. Baru, R. Moore, A. Rajasekar, and M. Wan, “The SDSC Storage Resource Broker,” in *ASCON '98 Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=783165 6

[108] "kopal," 2007. [Online]. Available: http://kopal.langzeitarchivierung.de 6

[109] E. Oltmans, R. van Diessen, and H. van Wijngaarden, "Preservation functionality in a digital archive," *Proceedings of the 2004 joint ACM/IEEE conference on Digital libraries - JCDL '04*, p. 279, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=996350.996416 6

[110] D. Giaretta, "The CASPAR Approach to Digital Preservation," *International Journal of Digital Curation*, vol. 2, no. 1, pp. 112–121, 2007. [Online]. Available: http://ijdc.net/index.php/ijdc/article/view/29 6

[111] A. Farquhar and H. Hockx-Yu, "Planets: Integrated Services for Digital Preservation," *The International Journal of Digital Curation*, vol. 2, no. 2, pp. 88–99, 2007. [Online]. Available: http://www.planets-project.eu/docs/papers/Farquhar_PlanetsIntegratedServices.pdf 6

[112] C. Becker, H. Kulovits, A. Rauber, and H. Hofman, "Plato: a service oriented decision support system for preservation planning," *JCDL '08: Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1378889.1378954 6

[113] H. Kulovits, C. Becker, and M. Kraxner, "Plato: A Preservation Planning Tool Integrating Preservation Action Services," in *Research and Advanced Technology for Digital Libraries*. Springer, LNCS Volume 5173, 2008, pp. 413–414. [Online]. Available: http://www.springerlink.com/index/317404U785H571TG.pdf 6

[114] SHAMAN-Consortium, "WP2.D2.3 Specification of the SHAMAN reference architecture," SHAMAN, Tech. Rep., 2009. [Online]. Available: http://shaman-ip.eu/sites/default/files/SHAMAN_D2.3-SpecificationReferenceArchitecture.pdf 6

[115] A. Rajasekar, R. Moore, C.-Y. Hou, C. a. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, *iRODS Primer: Integrated Rule-Oriented Data System*, Jan. 2010, vol. 2, no. 1. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00233ED1V01Y200912ICR012 6

[116] P. Wittek and S. Darányi, "Digital Preservation in Grids and Clouds: A Middleware Approach," *Journal of Grid Computing*, vol. 10, no. 1, pp. 133–149, Mar. 2012. [Online]. Available: http://www.springerlink.com/index/10.1007/s10723-012-9206-7 6

[117] S. Strodl, P. Petrov, M. Greifeneder, and A. Rauber, "Automating logical preservation for small institutions with Hoppla," *Research and Advanced Technology for Digital Libraries*, pp. 124–135, 2010. [Online]. Available: http://www.springerlink.com/index/WX626W572T4218QN.pdf 6

[118] S. R. Kruk and B. McDaniel, Eds., *Semantic Digital Libraries*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-540-85434-0 7.2

[119] J. E. Beaudoin, "Context and Its Role in the Digital Preservation of Cultural Objects," *D-Lib Magazine*, vol. 18, no. 11/12, 2012. [Online]. Available: http://dlib.org/dlib/november12/beaudoin/11beaudoin1.print.html 7.2

[120] A. Rauber, "Digital Preservation in Data-Driven Science: On the Importance of Process Capture, Preservation and Validation," in *Proceedings of the 2nd International Workshop on Semantic Digital Archives (SDA 2012)*, no. Sda, 2012, pp. 7–17. [Online]. Available: http://ceur-ws.org/Vol-912/paper0.pdf 7.2

[121] M. Mikelakis and C. Papatheodorou, "An ontology-based model for preservation workflows," in *8th International Conferece on Preservation of Digital Objects*, 2011. [Online]. Available: http://www.ionio.gr/~papatheodor/papers/ipres2012.pdf 7.2

Index

IVAN SUBOTIC

Ebnetstrasse 18 4106 Therwil Switzerland ibs@subotic.ch

EXPERIENCE

UNIVERSITY OF BASEL, IMAGING AND MEDIA LAB 08 / 2011 - 04 / 2013

Research Assistant

- Software engineering work in the field of web image databases
- Research and software engineering work in the filed of storing digital data on microfilm

INTESYS GMBH 10 / 2009 - 11 / 2011

IT Consultant

- IT Consulting with focus on proAlpha ERP solution including project management, development, implementation, testing and training tasks.
- Project management, development and implementation of solutions in the Web with WaveMaker and for the iPhone and iPad.

GLOBAL NETWORKS SWITZERLAND AG 08 / 2007 - 10 / 2009

Project Manager Network Services and Applications

Designing and implementing services and applications for GSM networks.

UNIVERSITY OF BASEL / IMAGING AND MEDIA LAB 11 / 2004 - 09 / 2008

Research Assistant

Working on the DISTARNET (DISTributed ARchival NETwork) Project funded by the SNF (Swiss National Science Fund)

UNIVERSITY OF BASEL / IMAGING AND MEDIA LAB 02 / 2002 - 10 / 2004

Software Engineer

- Working on the KTI project “Digitales Kino” (Digital Film) where I programed plugins for the film-compositing software “Rayz”.
- Design and implementation of a word-processor for hieroglyphs for the Egyptological Institutes of the University of Basel.

ASETRA GROUP AG 12 / 1999 - 05 / 2002

System Engineer

Design and implementation of projects involving Lotus Notes/Domino, Windows NT/2000 Client and Server, Tru64Unix and Compaq Alpha Server, Legato Networker Backup Solution

EDUCATION

UNIVERSITY OF BASEL 2007 - 2013

Doctor of Philosophy (PhD) , Computer Science in the group of Prof. Dr. Heiko Schuldt

UNIVERSITY OF BASEL 1992 - 2004

Master's degree , Business and Economics