Preservable Digital Objects as Building Blocks of Digital Libraries (PDO)

Michael Day[1], Maria Guercio[2], Volker Herrmann[3], Hans Hofman[4], Seamus Ross[5], Stefan Strathmann[6], Stephan Strodl[7], Manfred Thaller[3]

[1] UKOLN, University of Bath, Bath, BA2 7AY, United Kingdom
m.day@ukoln.ac.uk

[2] ISTBAL, Universita Degli Studi Di Urbino, Via Piano Santa Lucia 6, 61029 Urbino (PU), Italy
m.guercio@mclink.it

[3] HKI, Universität zu Köln, Albertus-Magnus-Platz, 50923 Köln, Germany
{herrmanv, manfred.thaller}@uni-koeln.de

[4] Nationaal Archief, Prins Willem Alexanderhof 20, 2595 BE Den Haag, Netherlands
hans.hofman@nationaalarchief.nl

[5] HATII, University of Glasgow, 11 University Gardens, Glasgow, G12 8QQ, United Kingdom
S.Ross@hatii.arts.gla.ac.uk

[6] Niedersächsische Staats- und Universitätsbibliothek Göttingen, Papendiek 14, 37073 Göttingen, Germany
strathmann@sub.uni-goettingen.de

[7] IfS, Technische Universität Wien, Favoritenstr. 9-11, 1040 Wien, Austria
strodl@ifs.tuwien.ac.at

Abstract. Digital libraries are traditionally seen as complex systems, which deal with various data objects. We propose to turn this view around. That is, we propose to define the functional capabilities of digital objects which we intend to administer in a digital library first and consider the digital library to be the sum of the functionalities required by these individual objects. We then describe digital objects, the methods of which have from the beginning be defined to fit the requirements of the ongoing discussions on preservation requirements. That is, any digital library, which is build from objects as the described ones, would implicitly be prepared for long term preservation.[1]

1 Introductory remarks

Preserving digital libraries is usually seen as an aspect of the interaction of the various parts of the system as a whole. While this approach lends itself easily to the design of overall engineering solutions, it leads relatively easily to complex systems (as e.g. emphasized by [2]) which are hard to analyse and where it is particularly difficult to identify what exactly the capabilities are, which are needed to allow the inclusion of an additional class of digital objects. Particularly so, when it comes to the requirements for long term preservation of these objects.

We explore in the following paper, therefore, a different approach and focus on the digital object itself, not the system in which it is administered. From this approach we expect to get a complete list of the functionalities, each digital object has to provide to fit easily into a preservation aware digital library system.

For that purpose, as will be derived further down, we assume that a digital library can be seen as a set of four packages of functionality, which form the environment, in which the various digital objects exist: (a) A package which encapsulates the capabilities to ingest the objects into an environment, which in itself can be completely transient, (b) a package bundling the capabilities, which are needed to allow the user (ignoring the differences between classes of users at this stage) to access the objects, (c) a package of the tools and capabilities

[1] This paper is part of the results of a workshop of the Preservation Cluster of the DELOS project at the Oxford Internet Institute, February 13th - Thursday 16th 2006.

needed to deploy the objects and, (d) a package of preservation functionality which controls all actions, which are needed to preserve the objects beyond the lifespan of the components of the packages (a) – (c): these serve the objects when they first are ingested into the system. A design for such a preservation package has already been described in a submission to the Preservation Cluster of DELOS (Herrmann & Thaller, 2005: [4]).

The relationship between the packages and the abstract objects described below is as follows. The abstract operations required to handle the individual objects are implemented as part of the respective packages, when a digital library is being built: The design of the individual objects is responsible for providing the technical specifications, which describe in detail, which requirements the packages have to fulfil, to handle this specific class of an object. On a practical level we can envisage these 'preservable objects' as wrappers, which bundle an external object - say a PDF document - together with the metadata needed for the various services which are using that document. Extending a digital library to serve a new class of external objects would consist of the introduction of such components for the individual packages, as are needed. So the definition of a preservable object can be seen as an abstract definition of all the methods for which implementations have to be provided, to fit it into a digital library. (We consider this *not* as new model for the formal description of a digital library, are explicitly not contradicting [3]. We propose simply a new point of view, which, however, may make the engineering of solutions markedly easier.)

2 Characteristics of a Digital Object to be Kept in a Digital Library

Digital objects are the core units a digital library deals with. While being used day by day, such an object may simply be an abstraction, where e.g. the data of a book is a set of files, while the metadata exist in a data base and the connection between them is described by some addressing scheme. We propose to recognise that this is a clearly different distinct state of such an object, that the one, where it is preserved 'for ever' on some storage medium, which may not allow immediate use and require to keep the separate components in one physical unit, to ease guaranteeing e.g. consistent migration[2]. Which are the basic attributes of a preservable digital object?

It is assumed that a digital object is only preservable if it can be described by the following attributes:

1. It is assumed that a digital object is *always* associated with *metadata* .
   We differentiate four types of metadata:
   (a) *Core* metadata are metadata in the traditional context of library information. (Descriptive and semantic metadata; up to a degree technical metadata, though, in accordance with general preservation assumptions, we keep them small in size.)
   (b) *Context* metadata describe the relationship of the object to other objects. This is a generalisation of the concept of structural metadata, which is crucial to extend the concept of digital libraries to the requirements of stake holders in the archival world.
   (c) *History* metadata describe the decisions about an object and the migrations it has undergone during its lifespan within the preservation environment.(Cf. [7])
   (d) *Precognition* metadata describe all qualities of an object and all decisions about the policy towards that object, which are needed to submit it to automatic preservation actions triggered without explicit human intervention.
2. A digital object contains *data* that is described by that metadata.
   Data can be
   (a) digital information (e.g., an image file).
   (b) a procedure to access non- digital information (e.g. the instructions how to access a bundle of handwritten pages on a shelf).
   (c) NULL (an indication, that the system expects to associate data with the metadata received at a later stage *or* that it remembers that data have been associated with the metadata at some previous stage, have been intentionally deleted later, however) .

2 In [4], p. 3, we describe the transfer of an object from the transient to a persistent state and vice versa as a set of changes to the format of each separate component. In the design we describe here, we strongly recommend to implement the methods moving an object from transient to persistent stage in such a way, that in the persistent stage the smallest possible number of distinct files exists.

(d) technical properties (describing the needs of the data for preservation and deployment).

(e) the approximate size of the digital information connected with this object either via its data or via the objects it contains according to item 4 below. (To allow an estimate of preservation and deployment needs while the data attribute is still NULL, the system actions needed for handling the object later have however to be evaluated, analysed or otherwise anticipated).

3. It has a *persistent identity*.
4. It can be infinitely *recursive*, i.e. a digital object can be part of another digital object which can be part of another digital object and so on. (This is particularly important in archival environments, where metadata apply at many more different levels of hierarchical data organisation than in library systems in a more narrow sense.)

Thus, to be persistent, a PDO has to have an assigned metadata object, data that may have the value NULL and an identity that is not transient but persistent.

Fig. 1 shows the PDO with its relationship to metadata. The operations within the PDO class correspond to the life-cycle of a PDO.[3]

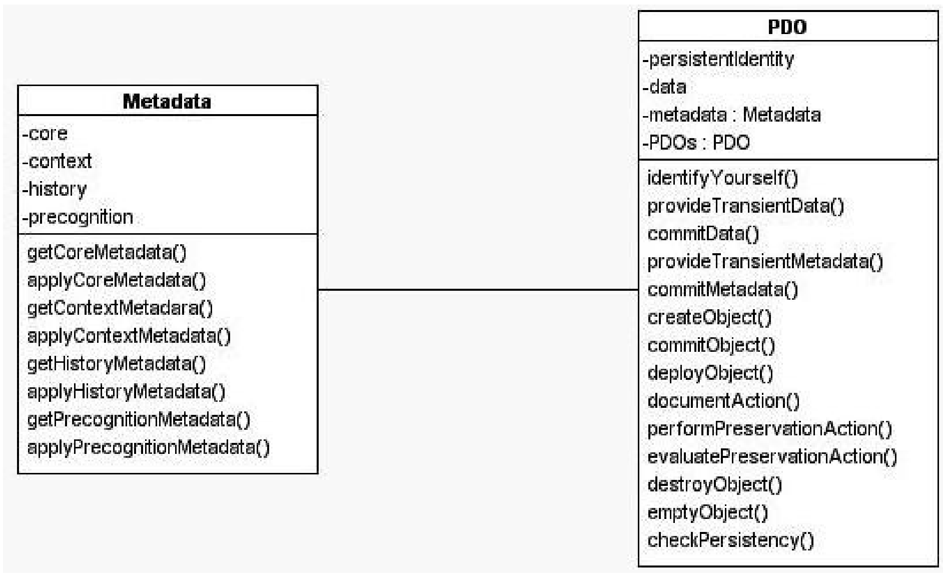

Fig. 1.

2 Relationship of Objects to Components of a Digital Library

The PDO is part of a complex digital library system. Within this complex system it mainly interacts with system components which reflect the basic operations on a PDO. The introduction of support for PDOs with a new class of external data objects consists of providing concrete implementations of all the abstract methods required.

There are four essential functional dimensions of a PDO: The PDO is

(a) *ingested*. Every PDO must be transferred into the digital library system. The process of ingesting the object includes a wide range of activities to be undertaken with the PDO.

(b) *used* by humans who e.g. need to select a PDO for display.

(c) *deployed*. The system needs to have access for the PDO for modifications, e.g. the request for the PDO needs to be managed. This involves operations on the PDO. 'Deployment' is a very general concept. It covers the actions needed to display an object on current visualization hardware as well as providing a bitstream of the object within a contemporary file format.

(d) *preserved*. This package includes all the functionality that comes along with the effort of preserving the PDO for long-term, e.g. migration of PDO components to current format.

3 The life-cycle of a PDO is discussed below.

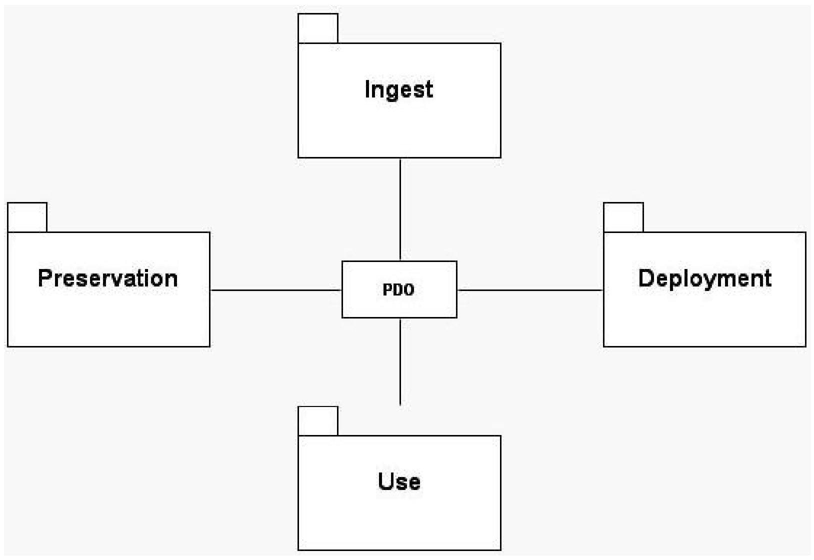

Fig. 2.

According to these four main functional dimensions of a PDO, a digital library system is assumed to consist of four core functional components which form the environment for the PDO (cf. Fig. 2).

3 Basic Functionalities of a PDO

The following passages elaborate the functionality of the PDO, necessary for fitting it into any overall digital library systems environment that claims to be aware of preservation, by analysing its life-cycle[4]. Therefore, this life-cycle is modelled as an UML state machine diagram with additional activity diagrams to provide a more detailed view on the subject. The analyse of the models will then lead us to a list of basic functionalities expressed as operations.

Due to both, space constraints as well as the context in which this paper has been prepared, we concentrate very much on those operations / methods, which are required for the purpose of preservation (which, of course, may influence also the behaviour of other packages than the preservation package proper). We assume, that the preservation package has the structure defined in [4]. A reader familiar with the layered architecture described by the Library of Congress in [6], 233 ff. can conceptualize our 'preservation package' as an 'intelligent repository', i.e., a design where some of the knowledge of the 'collection' level is in itself stored within the 'repository' level.

Fig. 3 shows the life-cycle of a PDO. State machine diagrams can also be described in terms of software application. Thus, for the description of the diagram we assume that the whole life-cycle of the PDO is carried out by any software application. According to that, from the view of software application and depending on the underlying programming language, all activities which occur in the diagram are called *operations* or *methods*.

A short remark concerning the terminology used further on: For example, the activity of creating an object is called 'createObject' within the state machine diagram. In terms of an application it could be written as 'createObject( )', indicating that there is an operation which is responsible for the act of creating an object. In this case the operation is expressed on a high level of abstraction, just telling that we have an operation responsible for the purpose of

[4] During the workshop quoted in note 1 this was based on internal design papers provided by some participants and checked for compatibility against [5].

creation but not telling which parameters and types the operation carries. The exact specifications of the full operations is part of the actual implementation of the functionality of the PDO within the digital library system in which it is intended to be embedded.

Thus, the particular functional properties of the PDO presented from here on further are all termed in that way, assigning a name on a high level but giving a preferably detailed description of the tasks of the functional part for future implementations.

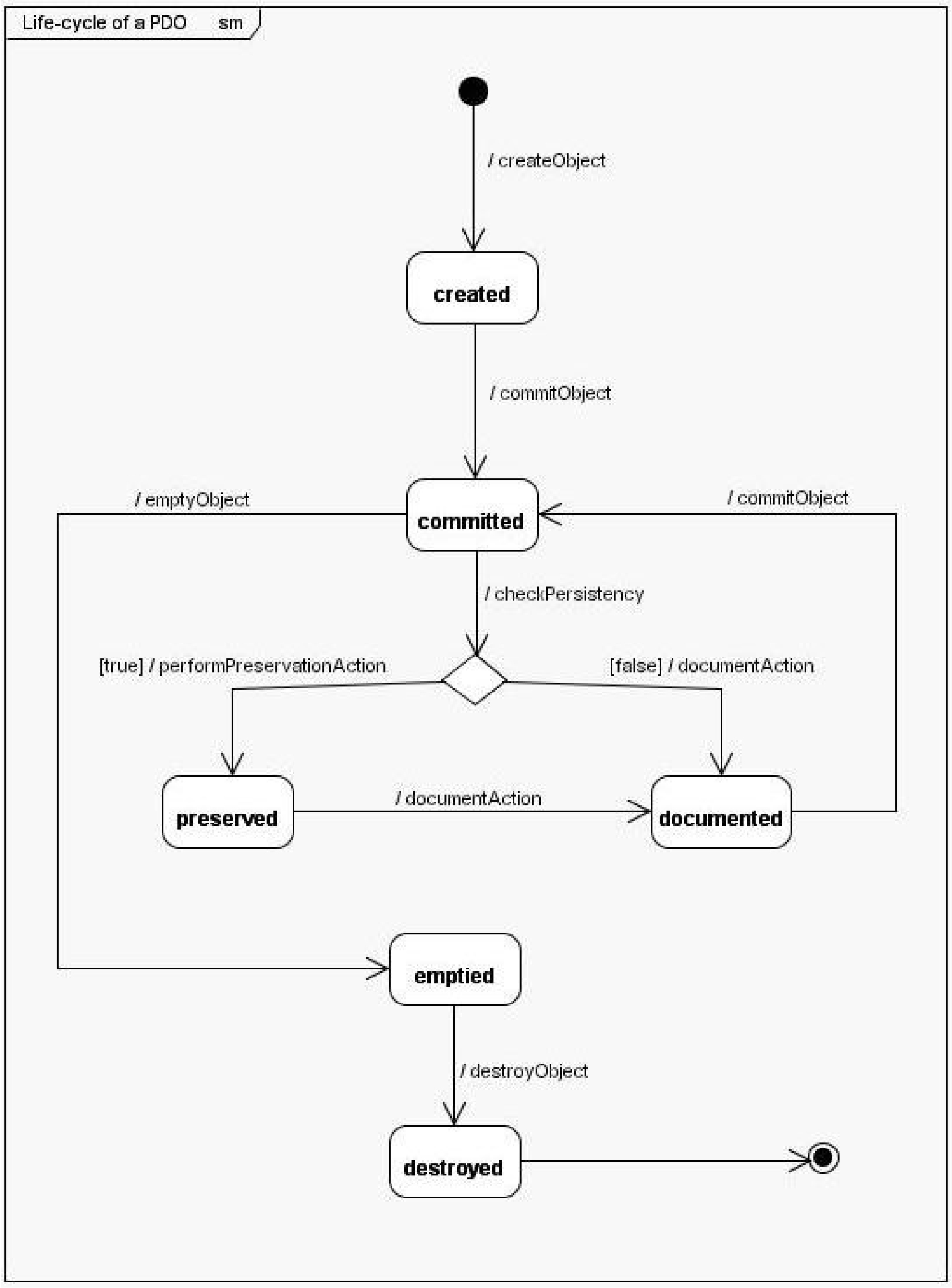

Fig. 3.

3.1 Functionalities relating to the Ingest and Preservation Package

Method: createObject( ). The first activity that takes place in the life of the PDO is its creation. Therefore, the first state a PDO can be in after its creation is the state of being *created*[5]. The according operation that reflects this functional part of the PDO is called ‘createObject( )’.

Let us regard the activity diagram in Fig. 4. It represents the core sub- functionalities within the overall process of creating the PDO.

When an object is created by the application it contains
(a) metadata
(b) *either* digital information *or* a procedure to access non digital- information plus technical properties
(c) an estimate of the size of the set of digital information eventually to be assigned to it,
(d) a preliminary, transient identity.

A workflow that contributes to this scenario of creating an object should have at least the functional components that are modelled through the activity diagram.

The act of generating metadata (on the applications level of description this could be a method: ‘generateMetadata( )’) leads to one part of the overall state of being created, the fact of containing metadata. The question on how to generate metadata is up to the specific implementation.

During the process of creation, the assignment of digital information to the PDO is also a fundamental part. In a simple scenario, this could be the assignment of binary data, e.g., an image file. Alternatively this can be a procedure for accessing non- digital information, e.g., instructions on how to access handwritten pages together with parameters that give information about the deployment or preservation of the entity. The decision node within the diagram reflects these circumstances as well as the operations.

In this stage of the PDO’s life- cycle, the actual set of digital information is not yet definite (the data can be NULL)[6]. Therefore, in case of assigning digital information, an estimate of the size of the set is necessary. E.g., this could be managed by an operation ‘estimateSize( )’.

Last but not least, creating the PDO also means assigning a temporary identity which is of transient nature until the PDO is transferred into persistent state. This is expressed through the activity ‘assign preliminary identity’ within the activity diagram.

Finally, the event of creation becomes part of the history metadata. This is modelled by using a rectangle box that represents an object, in this case the history metadata object, which is instantiated during the action of generating the history metadata[7] and passed over to the action of getting this data. This action is a functional part of the metadata object[8].

Method: commitObject( ). After the PDO is in the state of being created, it can be transferred to a new state that is named *committed*[9]. If, for any reason, the application stops before the PDO is committed, it goes out of existence.

The transition between the old state and the new state of being committed is from the applications view executed by an operation that is called ‘commitObject( )’[10]. The activity of commitment causes two further changes to the object: It receives a persistent identity and all of its components are transferred into persistent storage. From this point on, the PDO is fully persistent.

[5] Cf. Fig. 3.
[6] Cf. section 1 (sub- point 2.: data), where this issue is explained.
[7] for the whole activity of the creation of the object.
[8] Cf. Fig. 1.
[9] Cf. Fig. 3.
[10] Cf. Fig. 1.

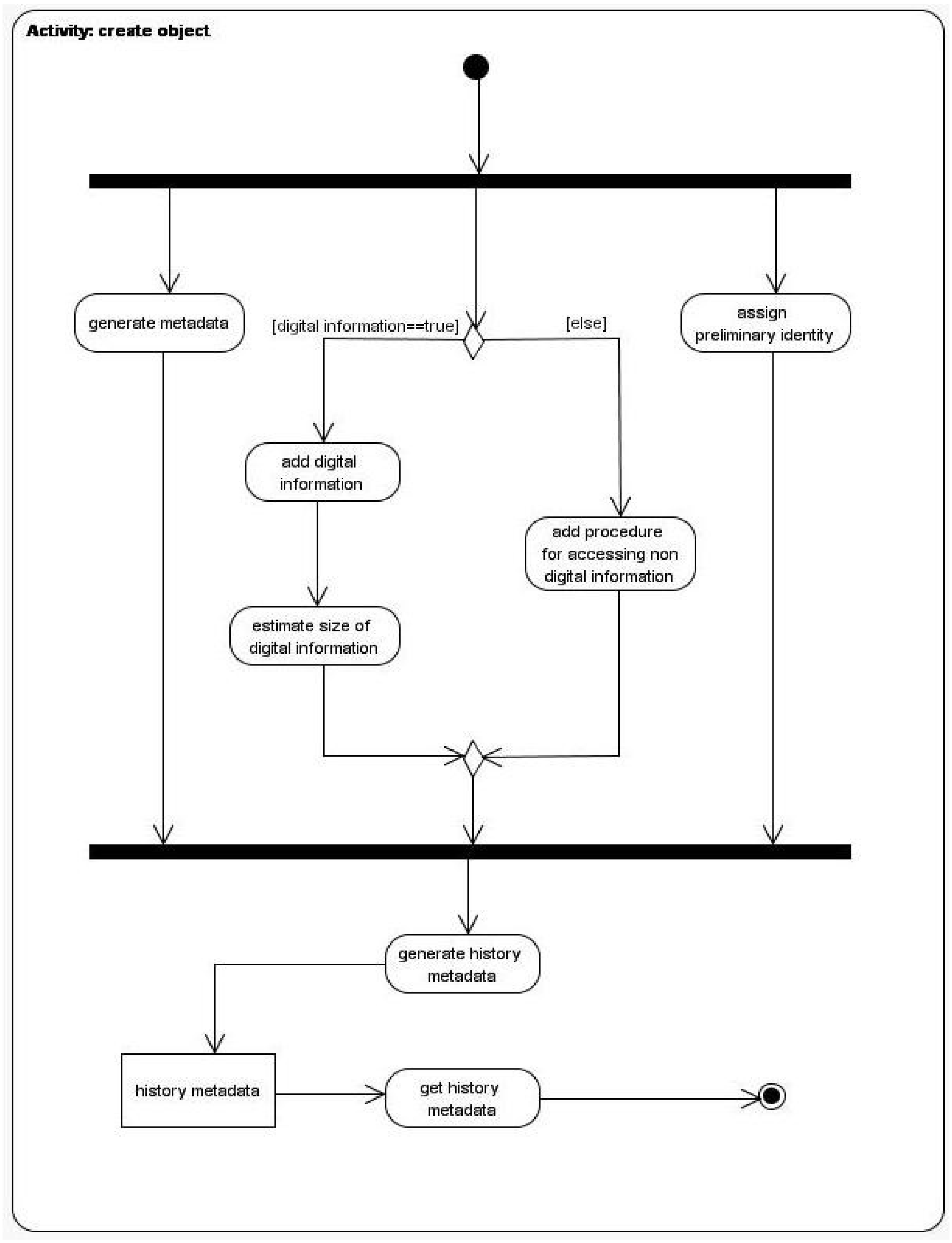
Activity: create object
generate metadata
[digital information==true]
[else]
assign preliminary identity
add digital information
add procedure for accessing non digital information
estimate size of digital information
generate history metadata
history metadata
get history metadata

Fig. 4.

The whole procedure of commitment of the PDO becomes also part of the history metadata.

Regarding Fig. 3 again, the commitment of the object appears a second time: The PDO does not remain in the state ‘documented’. A transition brings the PDO back to the state of being committed. This is the activity of (re)committing the PDO. Again, as a result, it is retransferred into persistent storage.

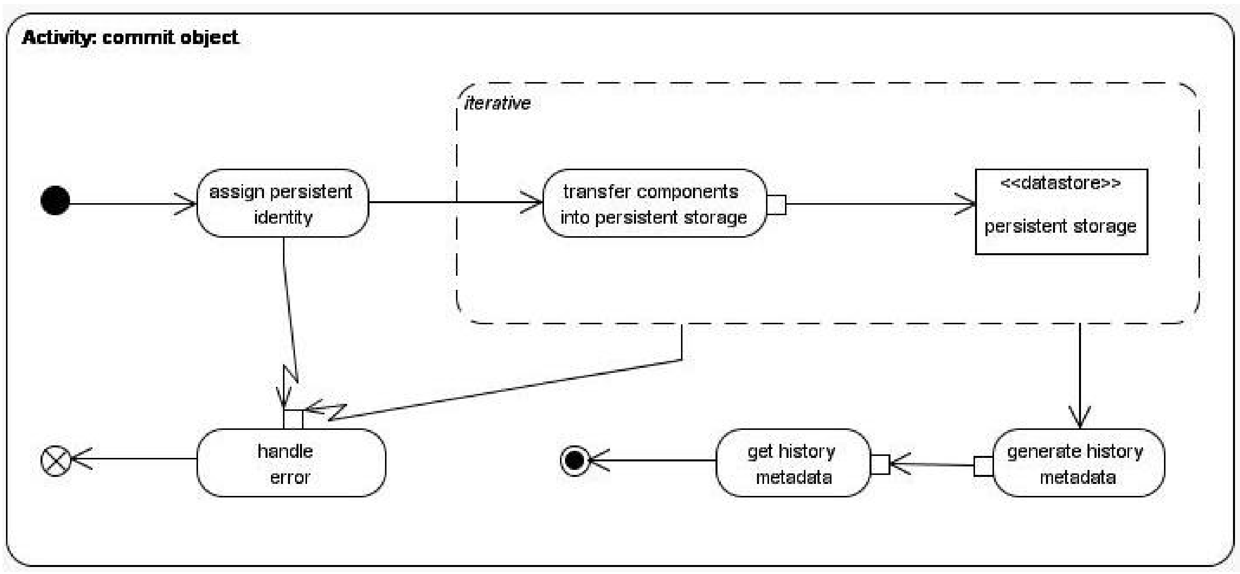

Fig. 5.

Fig. 5 shows the activity of committing the PDO. The action of transferring the components into the persistent storage is modelled as a expansion region with associated object nodes which are standing for the single components of the PDO that are iteratively passed to the persistent storage (modelled as an object of type 'datastore'). When all components are passed, the activity is successfully performed and the history metadata is finally generated. In case of failure, an exception handler is invoked (flashed arrow) that handles the error occurred (can also be the above mentioned application errors). The flow then ends at this point of the overall procedure.

Method: emptyObject( ). Let us regard Fig. 3 again. Being in the state of 'committed' the PDO can alternatively change its state anew. There are two possibilities of transition of the PDO: The first one, which can be regarded as the irregular one at this 'early' stage of the PDO's life-cycle, is a transition from the current state of being committed directly to the state of being *emptied*. The application's equivalent of the activity of emptying the object is an operation that is in similar called 'emptyObject( )'. When an object is emptied, the data are deleted from persistent storage and the value of the data attribute is set to 'NULL'[11]. Again, the event of emptying becomes part of the history metadata.

Method: destroyObject( ). Every life-cycle model ends with the death of the modelled entity. A PDO can also die. The operation of the application, logically equivalent to the activity of destroying the PDO, which causes this is 'destroyObject( )'. When an object is destroyed, it ceases irrevocably to exist and all components are deleted in permanent storage, *except* the core metadata plus a shortened object history, documenting its destruction. Therefore the state 'destroyed' is added to the state diagram of Fig. 3. The ultimate death of the PDO (the diagrams final state) is achieved when all related data is deleted from storage.

3.2 Functionalities within the preservation cycle.

Method: checkPersistency( ), performPreservationAction( ), documentAction( ). Because of the intention of long-term storage of PDOs, the regular way the PDO goes is a repetitive checking of the persistency of it. We call this *preservation cycle*. For example, it is assumed that the PDO will be checked on consistency of metadata after a certain span. Therefore, the transition between the states 'committed' and 'emptied' without being checked a single time in its life-cycle before, can be seen as irregular, while the transition that goes along with the activity of checking the PDO for its persistency can be regarded as the regular one that happens many times to the PDO. The ideal case in the life of the PDO would be the case of

[11] Cf. the remarks concerning data and its possible attribute NULL in section 1.

never reaching the state of being emptied or even being destroyed. This is only valid if we believe in the improbable case of a PDO that never dies - not even in a thousands of years.

Let us regard Fig. 3 again and additionally Fig. 6 in which the preservation cycle is modelled from the activity diagram's view, showing the fundamental workflow within the cycle.

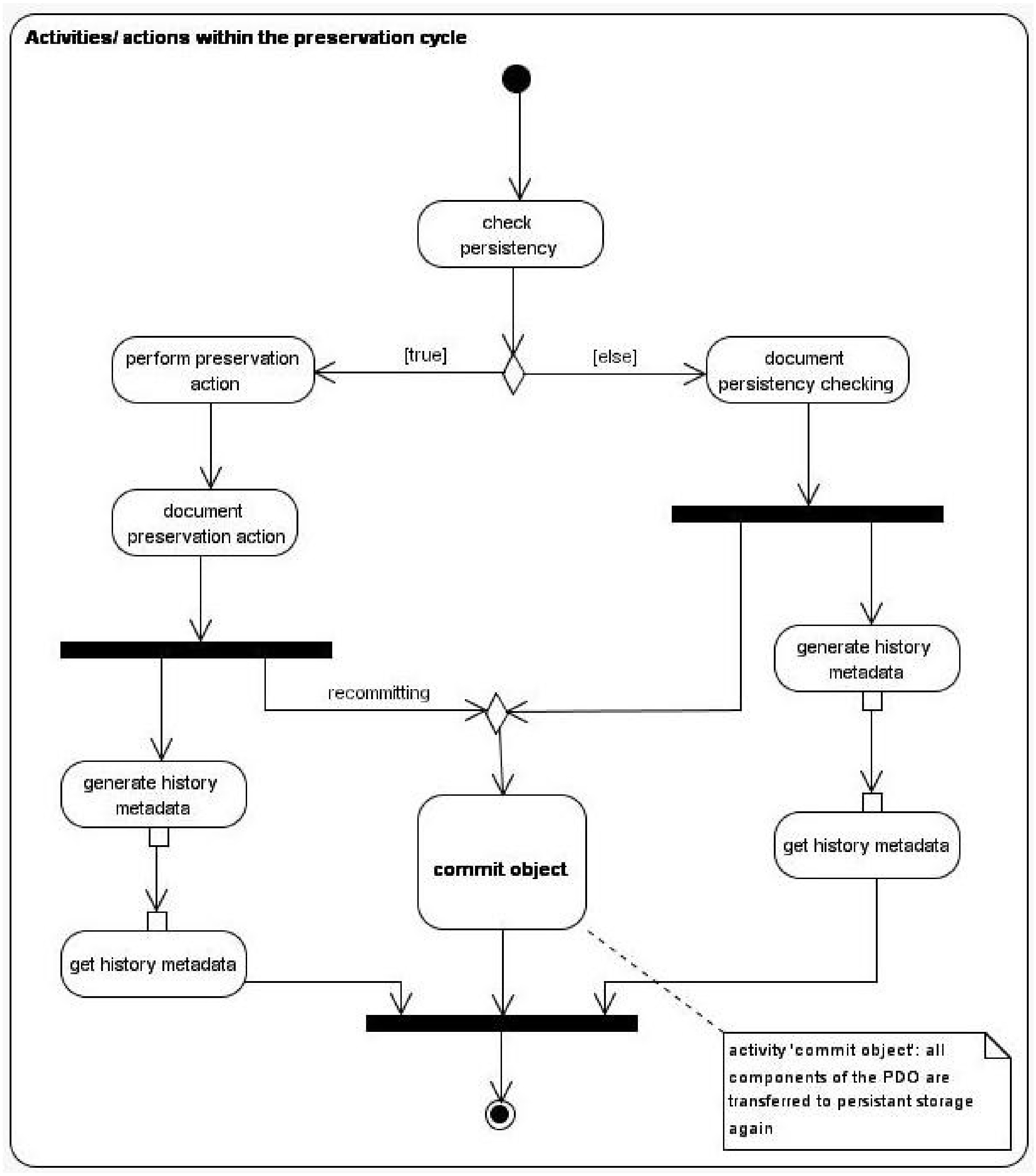
Activities/ actions within the preservation cycle
check persistency
[true]
[else]
perform preservation action
document persistency checking
document preservation action
recommitting
generate history metadata
commit object
generate history metadata
get history metadata
get history metadata
activity 'commit object': all components of the PDO are transferred to persistant storage again

Fig. 6.

In the preservation cycle, the application invokes an operation that checks the PDO for persistency ('checkPersistency( )'). It is assumed that there are exactly two alternatives which can follow on the checking of the PDO for persistency. Within the model presented here, the result of the checking of the PDO for persistency can be either 'true' or 'false', according to the return- type 'boolean' of the operation 'checkPersistency( )'.

If any deficiency in the persistency of the PDO is detected (return value 'true'),a preservation action on the PDO will be performed (activity/ operation 'performPreservationAction( )'). Then the PDO gets in thestate of being preserved. It is important to remark that all of the actions of the activity diagram in Fig. 6, are probably elaborated in a sophisticated design.

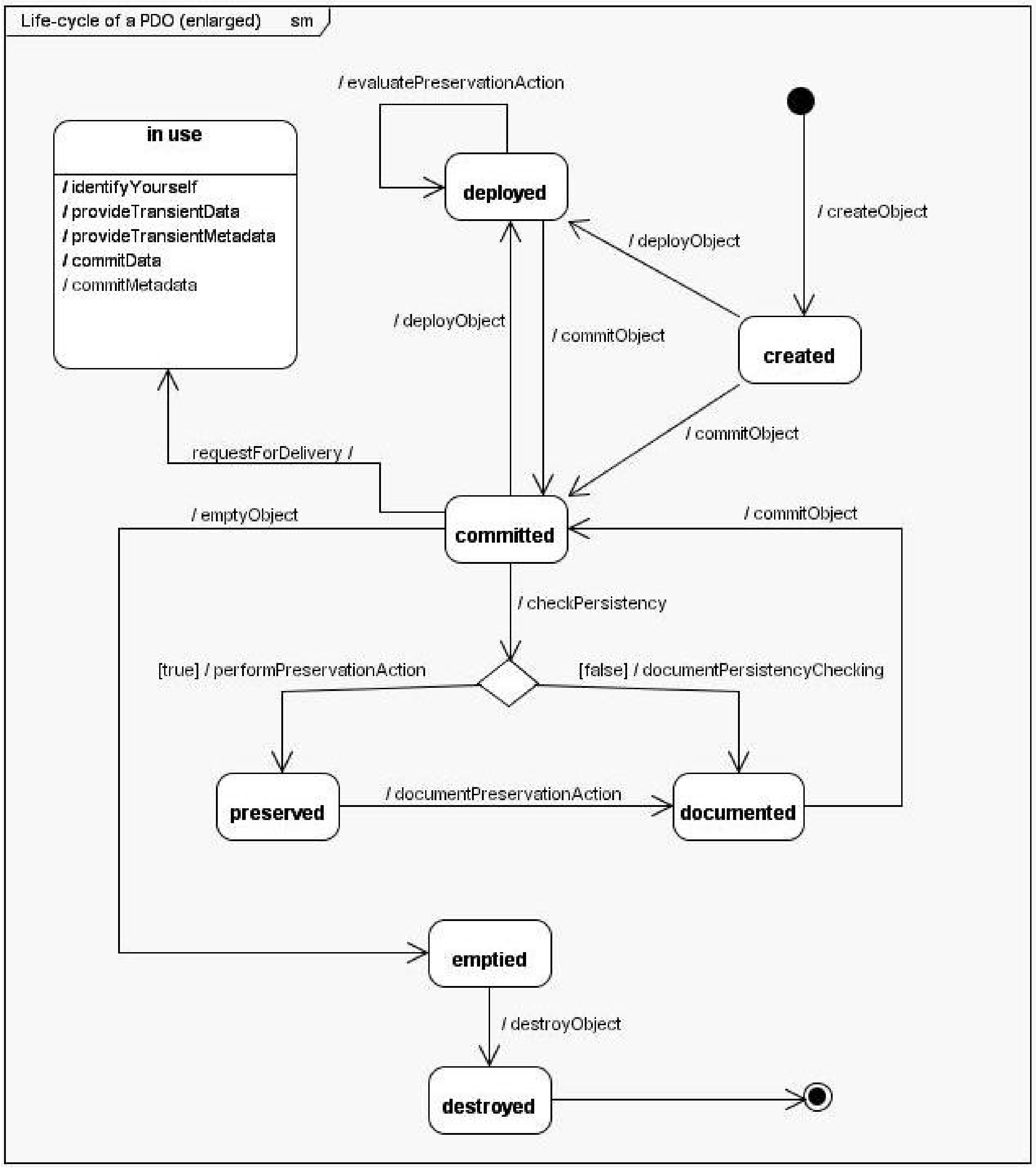

Fig. 7.

The actions then itself would become activities[12] (especially the performance of a preservation action[13]). Next to follow up is a transition that goes along with the activity of documenting the preservation action resulting in the state of being *documented*. This is the

12 In the UML, the term 'activity' is understood as a couple of actions.

13 Cf. [4], p. 6, where we have already modelled a more detailed scenario of a preservation workflow. We would like to make explicit, that a 'preservation action' can consist of triggering a migration to a more current file format as well as warning the system administrator of the need to provide an emulator for the deployment package. This approach is neutral; therefore, between the approaches at preservation differentiated, e.g. in [1].

task of an operation we term ‘documentObject( )’. For implementation purpose, this method will be specified. As a result, we then have at least two overwritten methods which would enlarge the basic functionality of documenting for the specific purposes, in this case at least for documenting the preservation action and for documenting the persistency checking.

The PDO can also change to state ‘documented’ directly. This is true for the boolean return value ‘false’. In that case, no deficiency in the persistency of the PDO is detected. The only activity/ action that has to take place is the documentation of the checking for persistency. Once more, both documentations become part of the history metadata.

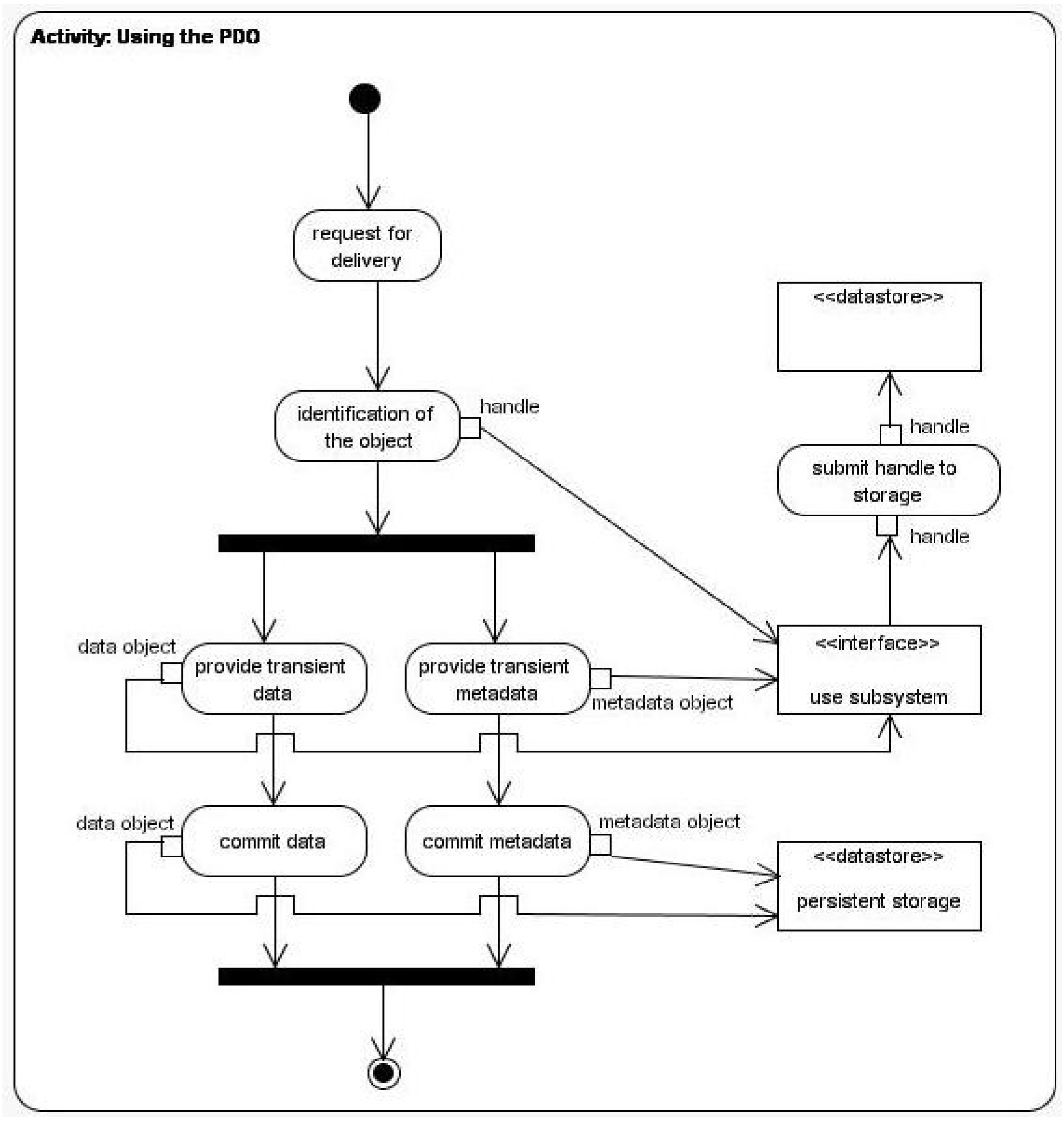

Fig. 8.

3.3 Functionalities relating to the Use and Deployment Packages

Fig. 7 presents an enlarged state machine diagram. Two new states have been inserted, the state of being *deployed* and the state *in use*. Both states, together with the functionality associated with these states, reflect the relationship of the PDO to the use and deployment packages introduced in section 1. In the following sections, the basic functionalities a PDO has to serve to cope with these parts of a digital library system are introduced.

Method: deployObject( ). It is assumed that between creation and commitment of an object the object, if appropriate, is deployed and preservation actions may be tested and evaluated.

The results of deployment[14] and evaluation (by program) are evaluated by the human archivist[15] responsible for the management process. As result of such a test- cycle, additional metadata and / or data will usually be added. If the evaluation is satisfactory and the PDO is complete, i.e., contains all data and metadata, it will be committed.

Operations for using the PDO. Between commitment and destruction of a PDO, the object can deliver its constituents to a use package / subsystem. This is modelled in Fig. 8.

Following operations are assumed to be basic for that purpose:

- ‘identifyYourself( )’ provides (e.g. a 'use' system / interface) with a handle that can be stored outside of the system and is guaranteed to remain accessible.
- ‘provideTransientMetadata( )’ provides an object allocated in memory, which can be modfied on its own.
- ‘provideTransientData( )’ also provides an object allocated in memory, which can be modified on its own.
- ‘commitData( )’ and ‘commitMetadata( )’ change the persistent presentation of these attributes. They may augment or replace part of the information represented by the PDO, do not change its identity, however.

In this part of the lifecycle of a PDO all activities which have been possible between creation and commitment are also possible. [16]

References

1. Davis, S.; Heslop, H.; Wilson, A. (2002): An Approach to the Preservation of Digital Records. [http://www.naa.gov.au/recordkeeping/er/digital_preservation/Green_Paper.pdf]

2. Endres, A.; Fellner, W. (2000): Digitale Bibliotheken. Heidelberg: D- Punkt.

3. Gonçalves, M. A.; Fox, E. A.; Watson, L. T.; Kipp, N. A. (2004) : Streams, structures, spaces, scenarios, societies (5S): a formal model for digital libraries. *ACM Transactions on Information Systems* , 22(2), 270- 312.

4. Herrmann, V.; Thaller, M. (2005): Integrating Preservation Aspects into the Design Process of Digital Libraries. (report submitted to the Delos NoE on Digital Libraries).
[http://www.dpc.delos.info/private/output/DELOS_WP6_d651_finalv3_5__cologne.pdf]

5. INTERPares project: Preservation task force report (2001).
[http://www.interpares.org/display_file.cfm?doc=ip1_ptf_report.pdf] and
[http://www.interpares.org/display_file.cfm?doc=ip1_ptf_model.pdf]

6. Library of Congress (2002): Preserving our Digital Heritage. Plan for the National Digital Information Infrastructure and Preservation Program.
[http://www.digitalpreservation.gov/about/ndiipp_plan.pdf]

7. McKemmish, S.; Acland, G.; Ward, N.; Reed, B. (1999): Describing records in context in the continuum: the Australian recordkeeping metadata scheme. *Archivaria* 48.

Note: All URLs valid as per 29th January 2007.

[14] The act of deployment contains provision for checking the integrity of the object, including, e.g., checks for viruses in the data.

[15] It is expected, that future developments will reduce the need for human intervention progressively.

[16] They may require a higher degree of authorisation. They also may be triggered by an explicit (human) request *or* as consequence of the precognition metadata stored with the object *or* as consequence of knowledge stored in the preservation memory of the preservation package / subsystem.