OMM: A Structure Model for Digital Object Memories (DOMe)

Jens Haupert
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
jens.haupert@dfki.de

***Abstract*—In this paper we address the research question, how an structuring model for digital object memories (DOMe) has to be designed. Primary goal is to identify and develop components and processes of an architecture concept particularly suited to represent, manage, and use digital object memories. In order to leverage acceptance and deployment of this novel technology, the envisioned infrastructure has to include tools for integration of new systems, and for migration with existing systems. Special requirements to object memories result from the heterogeneity of data in so-called open-loop scenarios. On the one hand, it has to be flexible enough to handle different data types. On the other hand, a simple and structured data access is required. Depending on the application scenario, the latter one needs to be complemented with concepts for a rights- and role-based access and version control.**

***Keywords*-Data models, Management, Data visualization, Standardization**

I. Introduction

Achievements of the last few years in the field of communication and information systems enable us to give physical objects a digital identity. This idea founded the *Internet of Things* (IoT) [1], [2]. In the first place a unique identification code was attached to real world objects that are machine-readable based on optical marker (e.g. bar-codes) or with radio transmission (e.g. RFID/NFC). This technology enables systems to automatically detect objects and process object data in different parts of the product life-cycle chain. But there is a trend to more sophisticated applications [3]. So called *Smart Labels* enable object-centric human-computer-interaction and machine-to-machine communication by providing a combination of unique identification and integrated sensors. Objects with such labels can monitor their condition along the entire life-cycle chain without the help of an instrumented and intelligent environment. Finally this development leads to the concept of the digital object memory (DOMe) that additionally adds a memory storage capability to smart labels [4]. With such a feature set, objects are ready to be used in so called open-loop scenarios, in which the life-cycle chain is not pre-defined at design time but dynamically linked. With digital memories objects can exchange data with all partners in the chain (e.g. manufacturer, supplier, logistics, retailer, and user). Hence the real-world objects were enriched by such digital memories. In this position paper we will introduce the Object Memory Model (OMM) as a foundation for frameworks for object memory solutions. The content of this position represents a executive summary based on the paper [5] and the book [6] edited by the author.

II. Object Memory Model

The architecture presented in this paper is intended to provide a data model to partition object memory data and a storage framework to persist and access this data.

A. Object Identification

The first step to attach a digital object memory to physical objects is to create a unique identification code (ID) for each instance of an object. This ID goes along with the physical object for the entire life-cycle chain and connects the object with its digital representation. Hence it is reasonable to attach this ID directly to the physical object to enable entities in the object's range to access the object's digital memory.

The DOMeMan framework uses a unique *Uniform Resource Locator* (URL) as memory ID. This has the advantage that such a URL not only represents the ID but also indicates the way of memory access (e.g. via a HTTP-connection through the world wide web). The framework supports two different approaches distinguished by the fact of having the memory directly attached to the physical object ("on-product", e.g. with a smart label) or by having the memory contend stored on a dedicated server ("off-product", e.g. in the web).

In the on-product case the object is a simple carrier of the read-only memory ID. This allows different solutions beginning with very cheap, printed, and optically readable 1D- or 2D-codes (e.g. barcodes, QR Codes, or Data Matrix Codes) and ranging to radio frequency identification (e.g. RFID or NFC) tags that can be accessed with dedicated readers or smart phones. However this approach requests an infrastructure to be available all the time to access the memory data.

In the off-product case at least a part or even the entire memory is located directly at the object, e.g. realized with small embedded systems. Hence no external infrastructure is needed to interact with the memory, with the disadvantage that such solutions are rather expensive and can be only used with upscale products. Notwithstanding the selected approach, a software component is necessary for interaction with the object's memory to parse and process the memory content.

B. Data Model

The foundation of the DOMeMan framework is the idea of storing information in an object memory. To achieve this goal a structuring data model is necessary to cover the needs of very heterogeneous data created in open-loop scenarios. The meta model created by the W3C Object Memory Model Incubator Group (OMM-XG) [7] and co-developed by the author provides a structuring element to partition the object memory information into several blocks, each with content of the same origin or nature (see Figure 1). The blocks consist of two parts: the payload representing the actual information and a set of meta data defining and describing the payload. Due to the heterogeneous data stored in object memories, access to such memories will lead to search operations (because the content is normally not known in advance) which can be performed only with the help of such meta data. The elements of the set are described in the following.

The first attribute called *ID* is a unique identification for each block represented as string. The next four attributes are intended for machine-to-machine (M2M) communication purposes. The *Namespace* is represented as URN and can be used to indicate that the payload has a defined content and a standardized type (e.g. "`urn:objectdata:pml`"). This allows for direct access to the payload, if the reader supports the namespace. *Format* is just the MIME-Type of the payload. *Subject* contains a tag cloud like structure to annotate the block payload with free text tags (e.g. "manual"), hierarchical text tags with a point as delimiter (e.g. "norms.din.a4") and ontology concepts (e.g. "http://s.org/o.owl#Color"). The *Type* attribute is a Dublin Core DCMI Type Vocabulary[1] Type. The following two attributes create a modification history for this block. *Creator* is a tuple of the entity that created the block and a corresponding timestamp. In addition *Contributor* is a list of tuples with entities that change the block and the corresponding timestamps. Finally the last two attributes are rather intended for human-computer-interactions (HCI). *Title* contains a human readable short title for this block and *Description* contains a longer textual description both with support for multiple languages.

In the case that the payload has a very large size and does not fit into the memory (e.g. located in an embedded system) or the data is redundant and used in many similar memories, the payload can be out-sourced. This is done by an additional *Link* attribute that indicated the source of the payload.

Furthermore each memory contains a header that includes the unique ID of this memory and an optional list of links to additional blocks (e.g. that are out-sourced due to space restrictions) and a table of contents (ToC) that provides the meta data information from all blocks to enable applications to get an overview of the memory content without the need to download and access all blocks.

[1]http://dublincore.org/documents/dcmi-type-vocabulary/#H7, [last access: 02/04/13]

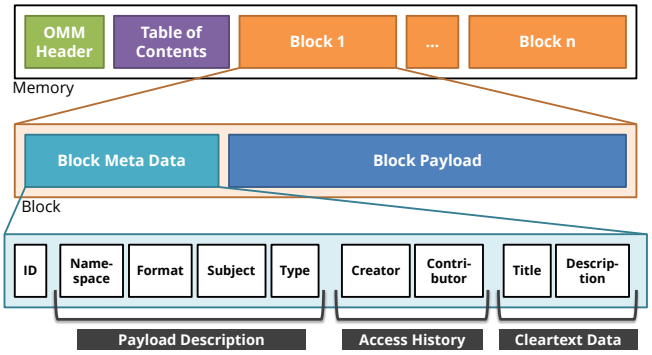

Figure 1. DOMeMan-Data Model for Object Memories based on W3C Object Memory Model Incubator Group (OMM-XG).

Generally the OMM does not provide a set of regulations for the block payload, so the users are free to store the information in the way they want or in the way they have defined with other partners. However the model includes three pre-defined blocks useful in several scenarios. The *OMM-ID Block* carries a list of identification codes combined with the corresponding string type and a validity timestamp or time-span. The *OMM-Structure Block* indicates relations of the physical object to other objects, without the need of additional formats. The user can use one or several of the following fixed relations: isConnectedWith, isPartOf, hasPart and isStoredIn. Each relation can also be combined with a validity time span. The *OMM-Key-Value Template* provides a container for an arbitrary amount of key-value-pairs that can be used in many use cases.

In addition we added a proposal for another three blocks (called OMM+) extending the OMM meta model. The *OMM+ Semantic Block* is an extension of the OMM-Structure Block and allows the definition of arbitrary relations similar to ontology relations (e.g. provided by RDF and OWL) relative to the physical object. Each relation consists of a triple (subject, predicate, and object) represented by uniform resource identifiers (URIs) combined with a validity statement. To indicate the physical object itself the URI `urn:omm:this` is used. The predicate can be use case specific or use common RDF/OWL object relations. It is also possible to include the OMM-Structure Block relations e.g. the isConnectedWith relation by using the URI `urn:omm:structure:isConnectedWith`. This block allows the definition of simple semantic statements that can be processed semantically (e.g. with a graph reasoner) or without a reasoner just compare the strings of the relation triple.

The *OMM+-Embedded Block* is intended to integrate an entire OMM-based memory into a specific block. In use cases where objects are physically integrated into other or larger systems it might be the case that the attached ID or the embedded system cannot be reached any longer. The

problem can be solved by copying the object's memory e.g. to the memory of the factory, so it can be accessed even if the object was integrated. A specific subject meta data attribute `primaryID.<ID of integ. object>` is used to indicate the embedded memory's ID without the need of extracting the memory itself.

C. Storage Infrastructure

Based on the described data model we developed a storage infrastructure for object memories with a two-tier approach. In the first place we created a software library (libOMM) serving as reference implementation of the object memory model for direct integration in Java- and .NET-based applications, to enable access to such local memories without any other functionality. Secondly we extend this library to a dedicated object memory server (OMS). The server is modularized into several components (see Figure 2) and can run with different combination of these modules depending on the intended application. Technically each module is implemented as servlet container and integrated with a common and shared web server component. It also allows the utilization of (parts of) the OMS on embedded systems, bringing its extended functionality even to on-product scenarios. The communication interface and the activity module are described in the following sections. The memories are handled by the storage module that was built on top of the mentioned libOMM and can handle memories in XML, RDFa and Microdata representations. For embedded scenarios an additional binary encoding is also available providing both a loss-less and a lossy mapping. The latter on can be valuable in case of restricted storage space. A memory designer can define the degree of lost (e.g. removed metadata or even removed blocks) that is represented by a mapping schema. There is also the possibility to restore deleted meta data information with such a schema.

The OMS can also provide a revision control mechanism. It creates a backup of the current memory state each time the memory is altered. Via the communication interfaces applications can also access older revisions and request a list of differences between two revisions. For debugging purposes it is also possible to reset the memory to an earlier revision.

Due to the idea of an open-loop data collector, the access to memories is not restricted generally, but the OMS is equipped with an role-based access module. The owner of a memory can restrict read and write operations for specific blocks and for entire memories. Three different approaches how to grant access to memories are available: passwords, certificates, and electronic ID cards. The simple approach uses a username and a password stored in a white-list containing all entities with access permissions. The more sophisticated approach utilizes digital certificates based on the ITU standard X.509 [8]. With certificates an additional way of restriction is possible: the certificate chain mode. This mode demands that an accessing entity uses a valid certificate and this certificate

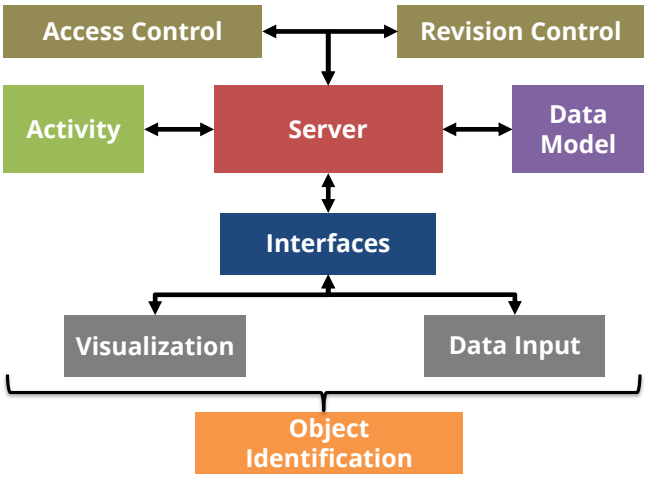

Figure 2. DOMeMan Architecture.

must provide a valid certificate chain with respect to the root certificate of the memory. The owner can select between two options. First the certificate must be directly signed by the owner or secondly a valid chain to the owner is sufficient. The latter option allows for certificate users to create their own child certificates and deploy them to other users of the memory. This approach lacks the risk that the user pool can be extended in an uncontrolled manner, but allows to distinguish between different "subcontractors" (each using individual certificates inherited from a accepted authority) without any administrative costs by the memory owner. Finally we created a prototype application based on the new German ID card (nPA). This card provides an electronic identification (eID) mechanism to create a unique but anonymous and application dependent ID for each card. This can be utilized to restrict memory access to such eIDs [9].

To increase the level of security the system also prevents the possibility of creating plagiarism by duplicating memories. If a server based approach is used, the given API allows only block-based memory access, so each block has to be copied to another fake memory, but these "new" blocks contain different creator information than the origin. If a solution based on RFID-tags is used, the framework can add an additional hash value incorporating the entire memory and the tag ID. A simple copy of the tag data breaks this hash value, due to a different tag ID.

D. Communication Interfaces

For memory interaction tasks two different interfaces are available (see Figure 3). Applications can use a RESTful HTTP-interface to access, to supplement and to modify object memories. End users can alter memories with the built-in web-based HTML5 user interface that can be utilized within a standard browser on multiple different devices. As mentioned before each memory can be accessed with a unique URL that serves simultaneously as access point and as the object's

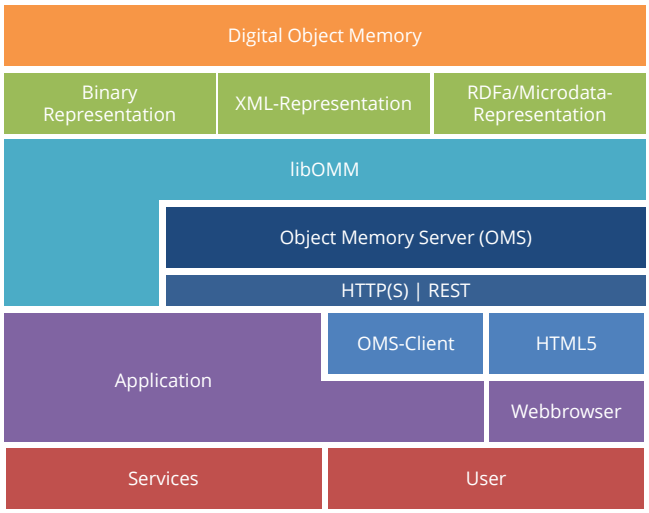

Figure 3. DOMeMan tools access hierarchy.

primary ID. This URL begins with the DNS name or IP address of the OMS and ends with the name of the memory. In between the caller indicates which module of the OMS should be triggered. This module can be set to 'st' for the RESTful storage interface or to 'web' for the HTML5 user interface. Further actions and commands deployed to this module are added to this generic URL part. In the following we use an exemplary memory that is stored on an OMS and accessible at the domain 'sampleoms.org' and the sample memory named 's_memory' this two URLs look like:

```
http://sampleoms.org/st/s_memory
http://sampleoms.org/web/s_memory
```

The RESTful interface, represented throughout the 'st' path, maps the functions and operations of the mentioned libOMM. The URL path `.../st/s_memory/block_ids/` retrieves the list of all blocks with their IDs that allows applications to access the entire memory. For data retrieval located in a block with unknown ID the function `.../st/s_memory/toc/` can be utilized to get the table of contents of this memory that is a compressed set of meta data from all blocks (e.g. large meta data like clear text description and tags are excluded). Finally a direct access to block meta data is possible, e.g. to access the creator of a block an application can use URL part `.../st/s_memory/<blockID>/meta/creator/` or `.../st/s_memory/<blockID>/payload/` to access or to change the block payload.

III. Conclusion and Outlook

In this paper we presented an infrastructure suited to represent, manage, and use digital object memories including tools for integration of new systems, and for migration with existing systems. Due to the special requirements resulting from the heterogeneity of data in so-called open-loop scenarios, we provided a flexible memory approach based on the object memory model (OMM) and complemented this core with a set of additional infrastructure tools including capabilities like a rights- and role-based access and version control to leverage acceptance and deployment. Future work will focus on the extension of the activity component (including a visual logic code editor), a sophisticated binary representation (with lossless and lossy conversion), and the integration of secure pedigree information. The model has successfully passed a W3C incubator group that certified the significance of a standardization.

Acknowledgment

This research was funded in part by the German Federal Ministry of Education and Research under grant number grant 01IA08002 (project SemProM), 01IA11001 (project RES-COM) and 01IS12050 (project OMM++). The responsibility for this publication lies with the authors.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] H. Chaouchi, Ed., *The Internet of Things: Connecting Objects*. Hoboken, NJ, London: Wiley-ISTE, 2010.

[3] D. Uckelmann, M. Harrison, and F. Michahelles, Eds., *Architecting the Internet of Things*. Berlin: Springer Berlin / Heidelberg, 2011.

[4] W. Wahlster, A. Kröner, M. Schneider, and J. Baus, "Sharing memories of smart products and their consumers in instrumented environments," *it – Information Technology*, vol. 50, no. 1, pp. 45–50, 2008.

[5] J. Haupert, "DOMeMan: A framework for representation, management, and utilization of digital object memories," in *9th International Conference on Intelligent Environments (IE) 2013, 9th, July 18-19, Athen, Greece*. IEEE, 7 2013, pp. 84–91.

[6] J. Haupert, *DOMeMan: Repräsentation, Verwaltung und Nutzung von digitalen Objektgedächtnissen*, ser. Dissertationen zur Künstlichen Intelligenz (DISKI). Berlin: AKA, 2013, vol. 339.

[7] A. Kröner, J. Haupert, M. Seißler, B. Kiesel, B. Schennerlein, S. Horn, D. Schreiber, and R. Barthel, "Object memory modeling - W3C incubator group report," 2011, last Access: 05/15/14. [Online]. Available: http://www.w3.org/2005/Incubator/omm/XGR-omm-20111026/

[8] ITU-T Recommendation X.509 Version 3, "Information technology - open systems interconnection - the directory: Authentication framework," 1997.

[9] B. Brandherm, J. Haupert, A. Kröner, M. Schmitz, and F. Lehmann, "Roles and rights management concept with identification by electronic identity card," in *8th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom-2010), March 29 - April 2, Mannheim, Germany*. IEEE Computer Society, 2010, pp. 768–771.