# Conceptual service architecture to synchronise research data management services using machine-actionable data management plans

FILIP ZOUBEK, TU Wien, Wien, Austria
TOMASZ MIKSA, TU Wien, Wien, Austria and SBA Research gGmbH, Vienna, Austria
ANDREAS RAUBER, TU Wien, Wien, Austria

Researchers of all disciplines produce, share, and reuse data as part of everyday research. Most funders require them to manage and document their data using data management plans (DMPs). DMPs are often static documents that researchers create by answering questions in predefined templates at the beginning of the research and, therefore, may become outdated and obsolete as the project progresses. It is essential to keep the DMP up to date at all stages of the research lifecycle since numerous stakeholders and various services participate in data management that depend on information from them. In this paper, we propose a conceptual service architecture that uses machine-actionable data management plans to automate the exchange and synchronization of information between different semi-automated research data management (RDM) services acting on behalf of different stakeholders. To solve the stated problem, we analyze typical use cases in which the DMPs change and formulate requirements based on which we developed the conceptual architecture. We depict the designed architecture through a set of views, namely physical, development, logical, and process, using UML and BPMN representation that describe the processes required to synchronize DMP information among multiple services. We instantiate it by implementing a service that connects a data repository and a DMP tool. Thus, we evaluate to what extent the defined processes help in keeping DMP contents up to date and which criteria must be fulfilled to keep them highly automated. The result of the paper feeds into a larger discussion on streamlining interconnectivity and machine-actionability across planning, tracking, and assessing research phases. It also facilitates consensus building on enhancing the Research Data Alliance's recommendation for machine-actionable DMPs.

CCS Concepts: • **Applied computing** → **Enterprise data management**; *Business process management*; • **Information systems** → **Digital libraries and archives**; • **Computer systems organization** → Client-server architectures.

Additional Key Words and Phrases: Data management plan; machine-actionable data management plan; research data management services; integration; software architecture; integration service

## 1 Introduction

Research data management (RDM) is the process of handling data throughout its entire life cycle, encompassing data collection, organization, storage, preservation, and sharing [5] to ensure its FAIRness [1] and reproducibility [26]. Many stakeholders are part of this life cycle. Mostly researchers who work with data on a daily basis, but also data stewards, infrastructure providers, funders, publishers, ethics reviewers, legal experts, repository operators, and university management, to name just a few. They all can use systems developed to perform specific RDM tasks. For example, researchers and data stewards can use data management planning tools to create data management plans (DMPs) [7]. They can also use data repositories to store data for the long term and to share it

Authors' Contact Information: Filip Zoubek, TU Wien, Wien, Austria; e-mail: filizub@seznam.cz; Tomasz Miksa, TU Wien, Wien, Austria and SBA Research gGmbH, Vienna, Austria; e-mail: miksa@ifs.tuwien.ac.at; Andreas Rauber, TU Wien, Wien, Austria; e-mail: rauber@ifs.tuwien.ac.at.


ACM 2158-6578/2025/1-ART
https://doi.org/10.1145/3712014

with peers. Funders have systems to store information about past, ongoing, or future projects [3]. Institutions where researchers perform their research usually use so-called current research information systems (CRIS) for managing, planning, and monitoring research in terms of finances and human resources, or to keep track of the scientific output, e.g. published data and publications, and to link them to specific projects and funders [6] [29]. The OSTrails[1] project identified pathways on how different RDM services can interact with each other by exchanging information [21]. The pathways are based on community consultations, among others the IDCC24 workshop [24], and present existing and expected interactions.

All these RDM services contain isolated information on a specific aspect of research data management. This information is often duplicated between systems and can change multiple times as the research progresses, e.g. location of data has changed, a new license was assigned, etc. To allow stakeholders to make their decisions on always up-to-date information and to evade inconsistencies between systems storing overlapping information, we must ensure synchronization and consistency of information across all services within the specific RDM context.

The Research Data Alliance (RDA) produced a recommendation on machine-actionable DMPs (maDMPs) to represent traditional DMP information using a set of properties to allow for the reuse and exchange of information by RDM services [12]. The recommendation is implemented by most of the DMP tool providers to import or export maDMPs [17]. It is an important and necessary feature, but it is still similar to a unidirectional data flow, rather than a continuous information exchange among various RDM services. There is an enterprise architecture in which maDMPs are used to connect services and thus automate RDM tasks [13]. Yet, all of these integrations are on a service-to-service basis and require custom adaptations. We still lack a universal framework to allow for integrating information from individual RDM services so that information included in maDMPs is frequently updated and reused by RDM services.

In this paper, we describe the conceptual service architecture to synchronise RDM services using maDMPs. Based on the state-of-the-art analysis and a case study, we identify typical use cases and requirements that become a basis for the service design. Specifically, we introduce a custom profile of the RDA recommendation for maDMPs with modified constraints to better deal with evolving identifiers for various types of entities and to track their evolution over lifetime. We also design generic processes for accessing and modifying specific parts of maDMPs. Following the design science methodology and based on these developments, we design and implement an integration service. The service uses the modified version of the RDA recommendation on maDMPs and provides a set of functionalities to allow RDM services acting on behalf of various stakeholders to use and modify parts of maDMPs, but only those to which access was granted. The service tracks the changes made by these services. The information stored within the maDMPs acts as the single source of truth for all other RDM services and can be accessed using the REST API. We evaluate the degree to which the proposed service improves the automation of the maDMP modifications using functional test cases. While the paper focuses on the design of the integration service, its implementation in the real-world setting will require further agreements within the community. The paper streamlines all these discussions and can be used as a framework to advance consensus building. For this reason, it is especially relevant to all system engineers and product owners of RDM services, as well as anyone involved in automation of RDM.

The paper is structured into eight sections. Section 2 presents related work. Section 3 describes the results of the requirements engineering process. Section 4 discusses the custom profile for maDMPs. Section 5 describes the proposed service architecture. Section 6 describes the implemented proof-of-concept. Section 7 deals with the assessment of the solution using functional test cases and evaluation of the level of automation. Section 8 provides a conclusion and discusses future developments.

[1] https://ostrails.eu

## 2 Related work

We discuss existing DMP tools, the RDA recommendation on maDMPs that is central to our discussions, and results of other attempts to automate and interconnect RDM services.

The primary limitation of traditional data management plans (DMPs) is their static nature, which can become burdensome for researchers [23]. The quality of these DMPs is contingent upon when they are created and the researcher's precision and proficiency with RDM [14]. There exist numerous resources available to assist researchers in the creation of DMPs. DMPonline[2] and DMPtool[3] are specifically designed to aid researchers in satisfying the requirements set forth by funding organizations [22]. Additionally, the Data Stewardship Wizard[4] (DSW) can automatically generate text-based DMPs by utilizing predefined templates through a questionnaire-based approach [19]. When considering the output generated by these applications, all three tools currently allow for the export of DMPs into static formats like PDF, and some even support machine-readable formats such as JSON or XML. DMP tools hold a pivotal role in the management of research data, making them a primary focus of our integration efforts within the realm of RDM services.

The findings of a community workshop [23] propose a shift towards the adoption of maDMPs. This transition facilitates the seamless integration of data management services and allows institutions to evaluate their current and future resource needs. The Research Data Alliance (RDA) has issued a recommendation [16] on maDMPs, aiming to enable the exchange and utilization of information stored in DMPs by automated systems. This recommendation serves as an application profile that leverages established standards like DCAT and Dublin Core while extending them with customized fields and constraints. Its primary purpose is to support the integration and information exchange between RDM services used by various stakeholders in research. We employ this application profile with additional custom extensions and assumptions to synchronize information among RDM services. Our work is also in line with the ten principles for implementing maDMPs [15], e.g. we follow the common data model, have version control, and ensure machine-readability.

The hackathon [2], organized by the RDA, was held in 2020 and covered a variety of topics, including the integration of machine-actionable data management plans. The work [4] focused on the endeavor to use an application profile in JSON format to share metadata amongst popular DMP tools, notably DMPTool, DMPonline, DSW, easyDMP[5], and Haplo[6]. The authors of [27] attempt to import and export the maDMP from or to their developed software named OpenDMP[7]. Zimmer et al. [28] present an endeavor to align the criteria of the RDA DMP Common Standard in both directions with the DMP tool Roadmap[8] and the data repository Figshare[9].

Argos is a DMP tool that can be customized to integrate with Zenodo and other repositories [11]. To store data in Zenodo, obtain a DOI for a DMP, or release a new maDMP version, Argos remaps DMP info into specific metadata and sends it to Zenodo's API with an app token due to Zenodo's lack of maDMP API support.

In summary, the primary research focus has been on mapping the maDMP model to existing system models, with generally positive outcomes. Most organizations or teams can create a maDMP in one service and import it into another, with some automating this process through API calls. This paper aims to facilitate frequent data updates across different RDM services, ensuring that even minor changes are properly propagated across the entire institutional ecosystem.



[2] https://dmponline.dcc.ac.uk/
[3] https://dmptool.org/
[4] https://ds-wizard.org/
[5] https://www.sigma2.no/data-planning
[6] https://www.haplo.com/
[7] https://gitlab.eudat.eu/dmp/OpenAIRE-EUDAT-DMP-service-pilot
[8] https://github.com/DMPRoadmap/roadmap
[9] https://figshare.com/

## 3 Requirements Engineering

This section presents the requirements engineering that we conduct based on the literature review and a case study of the largest technical university in Austria. First, we define the actors for whom the integration service is relevant. Second, we formulate user stories that the service should support in order to meet the stakeholders' expectations. Third, we derive functional requirements based on the user stories. Fourth, we model these functional requirements into specific use cases that drive the design of the service. The methodology and terminology are in line with [10] [20].

### 3.1 Actors

We categorize actors into two primary groups: *users* and *administrators*. Researchers, funders, publishers, ethics reviewers, legal experts, repository operators, infrastructure providers, research support staff, or institutional administrators are involved in the process of DMP development [12] and should thus use the integration service to synchronize the DMP information among the RDM services. We classified this group of stakeholders as users.

The integration service must also facilitate the management of data related to RDM services, whether they are newly deployed, relocated, require updates, or, in some cases, are discontinued. This management responsibility falls solely on those individuals overseeing the RDM infrastructure. Therefore, we have introduced a second actor, referred to as administrators, encompassing only institutional administrators among the defined stakeholders.

### 3.2 User Stories

Based on the explored research domain and current state-of-the-art, we collect business requests that make sense from the perspective of the defined actors from Section 3.1. We formulate them into the following list of user stories:

**US1**: As a user, I want automatic synchronization of DMP information within one institutional ecosystem between the services that I use to manage research data after every modification I make in one of them.
**US2**: As a user, I want to have up-to-date DMP information within one institutional ecosystem, even if it is changed by other users in their RDM services.
**US3**: As a user, I want to be able to get older versions of DMP.
**US4**: As a user, I want to be able to find out who added, modified, or removed specific DMP information.
**US5**: As an administrator, I want to be able to prevent RDM services from changing information that is not within their scope.

### 3.3 Functional Requirements

We derive the list of the functional requirements from the formulated user stories (Section 3.2) in the context of the integration service and the maDMP recommendation.

Each maDMP consists of **maDMP objects**. Each object is a set of **maDMP properties** that have a common **object identifier**. For example, maDMP consists of multiple *Datasets*. Each dataset has an *identifier* and properties like *title, description*, etc. The identifier of the maDMP object allows us to distinguish different instances of these objects from each other. Many objects contain identifiers that can be reused for these purposes from the definition. For others, we were able to find a single property that would fulfill this functionality. Identifiers are not persistent, as they can be changed during the development of the maDMP. It can be demonstrated by an example where researchers have a dataset stored on a commercial cloud without a persistent identifier, which will be assigned at a later stage of the research. The RDA recommendation does not specify how to deal with changes in the maDMP object identifiers, e.g. when a dataset gets a new identifier when moved between services hosting them. The integration service must take such changes into account.

We identify the following requirements:

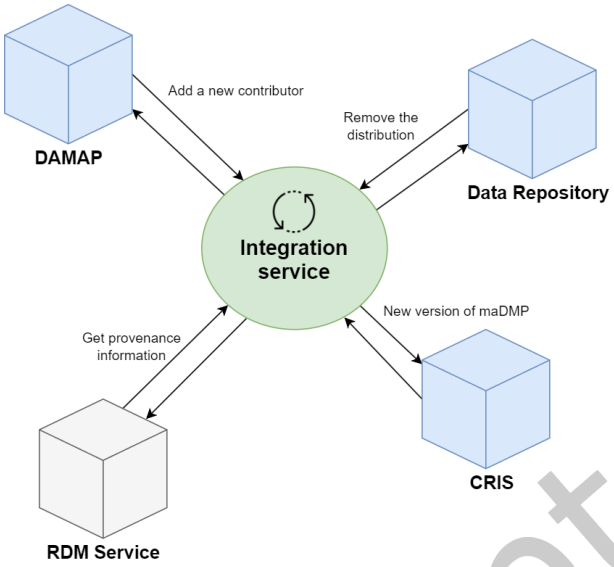


Fig. 1. Example of architecture using integration service to fulfill all functional requirements.

**FR1: Modify maDMP properties and objects**. The integration service will allow the user to modify DMP properties and objects, and synchronize the changes with RDM services within one institutional ecosystem using maDMPs.
**FR2: Modify the maDMP object identifier**. The integration service will allow the user to modify the maDMP object identifier and synchronize the change with RDM services within one institutional ecosystem.
**FR3: Remove the maDMP object**. The integration service will allow the user to delete the object from the maDMP and synchronize the change with RDM services within one institutional ecosystem.
**FR4: Get the maDMP**. The integration service will allow the user to obtain the current or older version of the maDMP.
**FR5: Get provenance information**. The integration service will allow the user to obtain the historical values of a specific maDMP property, including the provenance information.
**FR6: Act with the help of semi-automated RDM services**. The integration service will allow the user to act with the help of semi-automated RDM services. It means that users do not communicate directly with the integration service but use the RDM service to act on their behalf.
**FR7: Set RDM service rights**. The integration service will allow the administrator to set rights to modify maDMP properties for individual RDM services.

Figure 1 depicts an example of how the integration service deployed at a research-performing organisation synchronises information and manages updates among RDM services. There are four RDM services used in the example: *DMP tool* that adds a new contributor to the DMP, e.g. a new person responsible for implementing it; a *data repository* that updates a distribution section of the DMP after deleting data, e.g. when data violates terms and conditions of the repository; an *RDM service* fetching provenance information of a specific maDMP object, e.g. to resolve any conflicts with Scientific Knowledge Graphs; *CRIS* that consums all the modifications, e.g. to update researcher information profile.

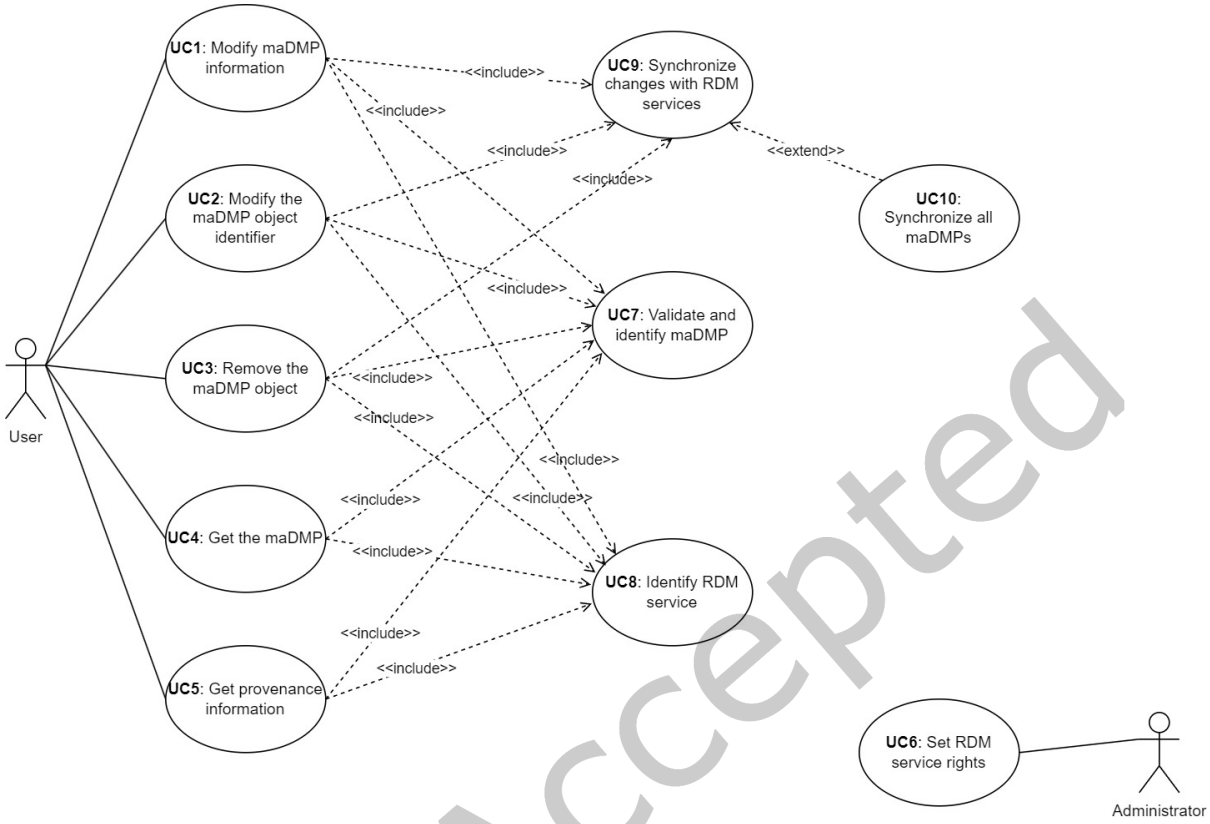


Fig. 2. A use case diagram showing the interaction between actors and individual use cases.



## 3.4 Use Cases

We capture the functional requirements from Section 3.3 by modeling the use cases that determine the boundaries of the integration service and show its individual functionality in more detail. Figure 2 depicts a use case diagram with the two identified actors. The administrator has only one use case, and that is to set the rights of RDM services to specific maDMP properties. On the other hand, the user interacts with five use cases, namely: modification of maDMP properties, modification of maDMP object identifier, deletion of maDMP object, retrieval of the previous maDMP version or the current one, and lastly retrieval of provenance information, i.e. the change history of a specific maDMP property with the actor that made it.

In addition, we identify several use cases that are included or extend the defined use cases. UC7 validates and identifies the maDMP, as all five use cases associated with the user work with the specific DMP in the form of the maDMP. This allows the integration service to recognize which particular maDMP is being worked with. UC8 identifies the RDM service, which is needed as, according to FR5 and FR6, users communicate through their RDM services and the integration service has to trace the provenance of the DMP information.

UC1, UC2, and UC3 are use cases that modify maDMP properties. It means that once the changes are successfully made, the other RDM services within the particular institutional ecosystem need to be informed about these modifications. Therefore, we identified UC9, which synchronizes new changes always for the particular maDMP. It

may happen that some RDM service cannot successfully process the new version of the maDMP or is unavailable, or a new one is added to the ecosystem and needs to be provided with with all maDMP instances. For these cases, we also modelled UC10, which synchronizes all registered maDMPs for the concrete RDM service.

## 4 Custom application profile for maDMPs

We also have to re-evaluate the application profile as a tool for transferring maDMP properties between RDM services within an integration service, which is a prerequisite to fulfill the FR1 from Section 3.3. There is a need to modify constraints, see Table 1, as not all RDM services can provide all mandatory properties, and the integration service needs to be able to identify each instance of the maDMP objects. Otherwise, we are unable to fulfill FR2 and FR3.

Table 1. The proposed changes in the property cardinalities of the application profile

| Object | Properties | Original cardinality | New cardinality |
|---|---|---|---|
| DMP | contact | 1 | 0..1 |
| | dataset | 1..n | 0..n |
| | title | 1 | 0..1 |
| | language | 1 | 0..1 |
| | ethical_issues_exist | 1 | 0..1 |
| dmp_id | type | 1 | 0..1 |
| Distribution | access_url | 0..1 | 1 |

We call the maDMP implementing the proposed constraints the **customized maDMP** in our design. This customized maDMP has only three mandatory properties within the object *dmp*, which is required in every version. These are the *created* property, which indicates when the maDMP was created and which can be used to identify the maDMP, the nested *identifier*, which can be used to identify the maDMP as well, and the *modified* property, which is used for versioning.

As part of the analysis, we also identify for each maDMP object a property that can be used for its identification within the customized maDMP, see Table 2. One could argue that the *dmp*, *contact*, *host*, and *grant id* objects are unique: *dmp* and *contact* within the maDMP, *host* within the *distribution*, and *grant id* within the *funding*. Your argument is sound. Only these objects do not require the identifier, and specific instances can be identified without one; however, from the standpoint of the integration service, we attempt to establish a consistent method for identifying the particular instance of the maDMP object that we found: the nested property that can be used to identify each instance of the customized maDMP object.

Table 2 shows the selected identifiers for each object in the customized maDMP. Lables: *dmp*, *cost*, *contributor*, *contact*, *project*, *funding*, *dataset*, *distribution*, *license*, *host*, *security and privacy*, *technical resource*, *metadata*, and *grant id* encapsulate a specific set of maDMP properties. Compared to the original schema, we classified *grant id*, which is part of the *funding*, as the object as well. As the *funding* object already has its own nested identifier, we considered it as a separate one that carries additional data.

## 5 Architecture

This section presents the architecture of the integration service based on the requirements from Section 3. We describe it using the 4+1 view model, which consists of four views: logical, process, development, and physical, complemented by scenarios [8]. We present only a small subset of considered scenarios due to length restrictions and skip the deployment view that would be specific to a particular implementation. Full documentation can

Table 2. The selected identifier of each object in the customized maDMP

| Object of the customized maDMP | Nested identifier |
|---|---|
| DMP | created, nested identifier |
| Project | title |
| Funding | nested identifier |
| Grant ID | identifier |
| Contact | nested identifier |
| Contributor | nested identifier |
| Cost | title |
| Dataset | nested identifier |
| Distribution | access_url |
| License | license_ref |
| Host | url |
| Security and Privacy | title |
| Technical Resource | name |
| Metadata | nested identifier |

be found in the Annex. For graphic design, we decided to use the Unified Modeling Language (UML) [9] with Business Process Models and Notation (BPMN) [18].

## 5.1 Process View

In this section, we present one of the designed processes, namely the business process for modifying the maDMP object identifier that is depicted in Figure 3. All others are in Appendix A. Specifically, you can find there processes for modifying maDMP properties, modifying the object identifier, deleting the object, identifying the maDMP, and the synchronization process of new information within the integration service.

The process (cf. Figure 3) starts when the integration service receives a message and identifies the RDM service and the maDMP. Subsequently, it is necessary to check a series of conditions, such as whether all necessary information was sent, whether the object has a changeable identifier, or whether the RDM service has rights to change the specific property. If they are met, the identifier of the object is changed along with all properties that depend on it. It then continues with changing the *modified* property and synchronizing new information with other RDM services.

In the process, we mention the condition check whether the object has the changeable identifier. We designed this step to limit the identifier change to maDMP objects where it makes sense as maDMP objects are identified by their properties. Table 3 displays the identifiers for each object extended by a column indicating if the identifier may change in the course of research but the meaning of the instance will remain.

In order to bring the issue closer to the reader, we give an example in the context of the *license* object. As *license* refers to the single particular specification, changing the identifier of this object affects the meaning of the entire instance. Switching the license reference from the MIT to the GNU GPL alters the entire meaning of the *license* object. Therefore, we prevented the *license* object from changing its identifier, and it is better if the user deletes the whole object and creates a new one. On the other hand, when a distribution's identification changes, it may be due to a change in the environment, but the *distribution* instance itself remains the same. This restriction can only be taken as a recommendation.

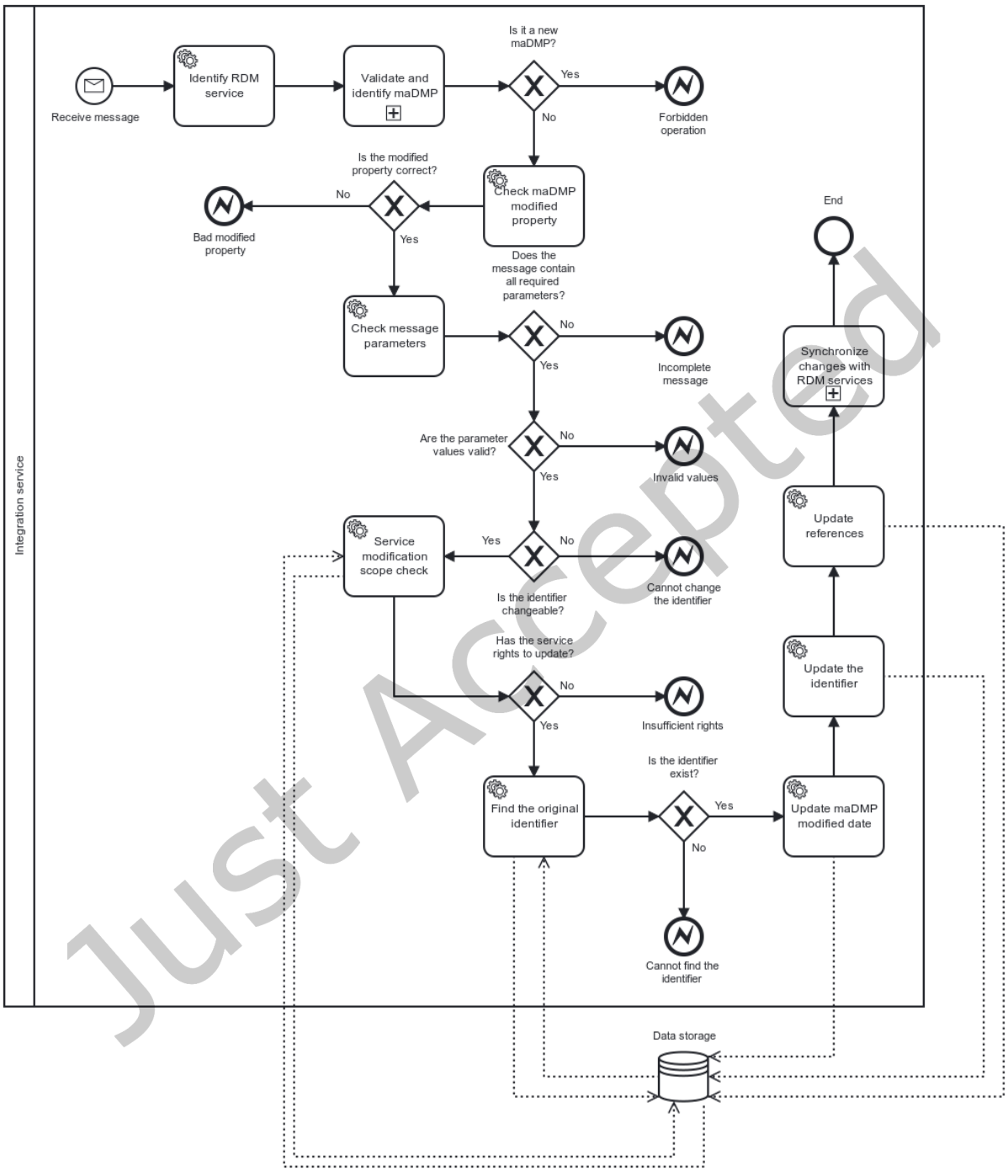




Fig. 3. Business process model for the use case 2.

Table 3. Recommended restrictions on changing object identifiers

| Object | Identifier | Changeable identifier |
|---|---|---|
| DMP | created, nested identifier | ✓ |
| Project | title | ✓ |
| Funding | nested identifier | ✓ |
| Grant id | identifier | ✓ |
| Contact | nested identifier | ✓ |
| Contributor | nested identifier | ✓ |
| Cost | title | ✓ |
| Dataset | nested identifier | ✓ |
| Distribution | access_url | ✓ |
| License | license_ref | |
| Host | url | |
| Security and Privacy | title | |
| Technical Resource | name | |
| Metadata | nested identifier | |

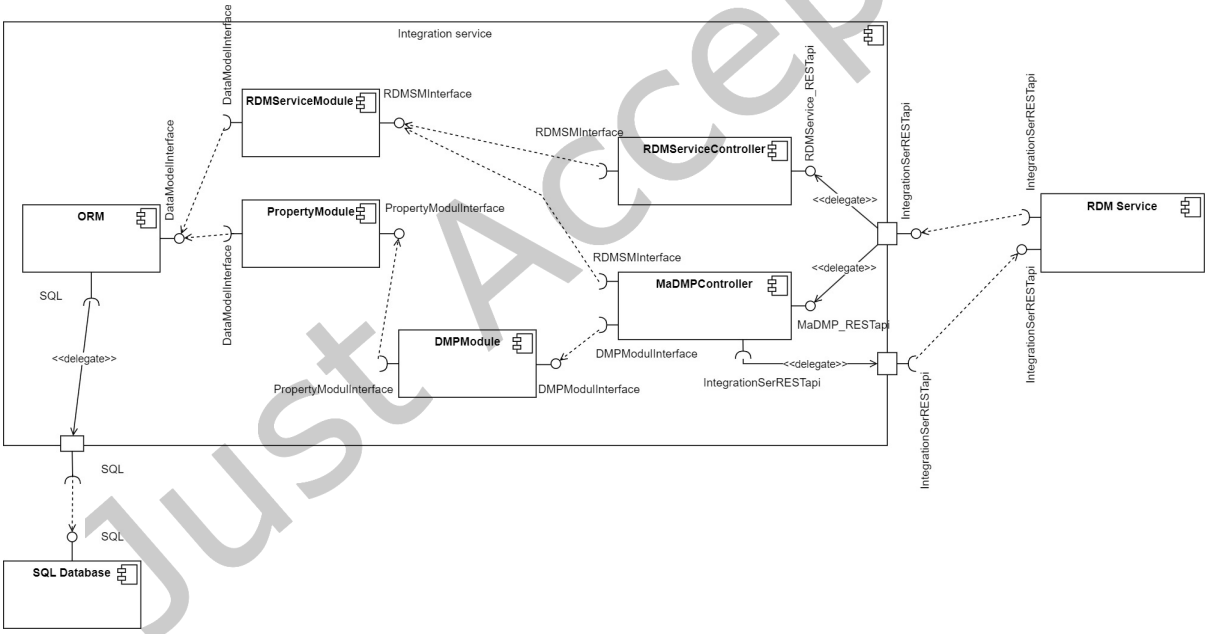


Fig. 4. A component diagram showing the individual components of the integration service.

## 5.2 Development View

The development view is depicted using the component diagram shown in Figure 4. The integration service itself consists of six components communicating with each other using clearly defined interfaces.

The system consists of key components:

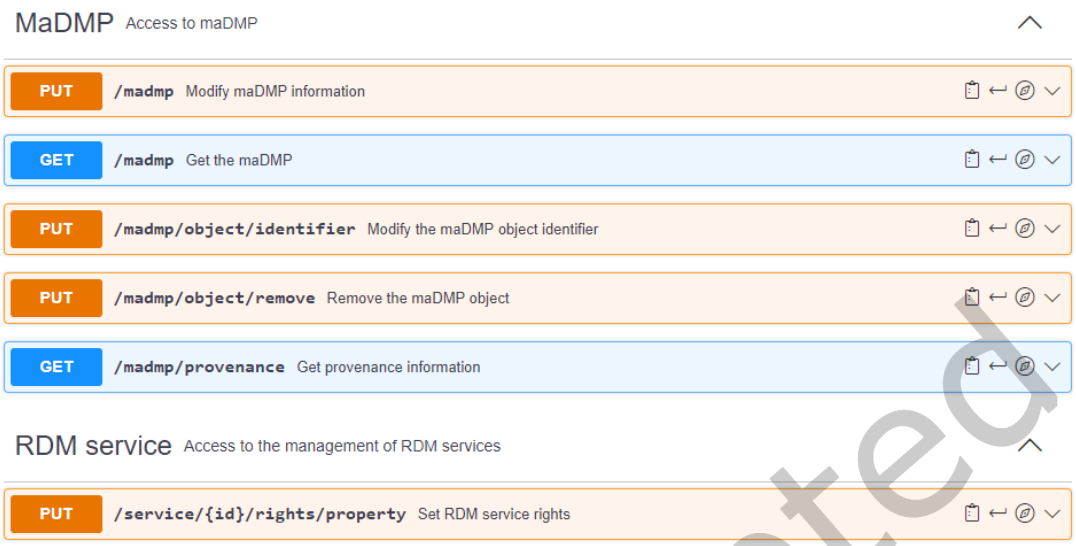


Fig. 5. Overview of methods defined for the REST API of the integration service.

- **RDMServiceController**: This component handles requests related to the management of RDM services, including setting rights for maDMP properties.
- **RDMServiceModule**: Responsible for the business logic of RDM services, it identifies the RDM service, retrieves instances, and manages their states.
- **MaDMPController**: This component processes requests for modifying maDMPs, including actions such as changing object identifiers, deleting objects, fetching current or previous maDMPs, and retrieving provenance information. It also ensures the synchronization of new information within RDM services.
- **DMPModul**e: This module contains the business logic for maDMP instances. It handles maDMP validation and identification, creates customized maDMPs with current values, and supports various maDMP use cases.
- **PropertyModule**: This component manages properties, which store information related to maDMPs. It is responsible for searching valid or outdated properties and creating or updating them.
- **ORM (Object-Relational Mapping)**: This component facilitates data storage by bridging the gap between a relational database and an object-oriented programming language.

Additionally, the system interacts with a database system for data storage through SQL queries. Communication between RDM services and the integration service is facilitated through a REST API, see Figure 5. This API has been designed and documented in the SwaggerHub application[10]. It comprises five endpoints for handling maDMPs and one endpoint for managing RDM services. Each endpoint is dedicated to a specific use case outlined in Section 3.4, representing interactions by users or administrators. The API specification includes parameters and body schema requirements for requests, along with defined responses that RDM services should appropriately handle to ensure data synchronization in case of errors.

[10]https://app.swaggerhub.com/apis/e11938258/InDMP/1.0.0

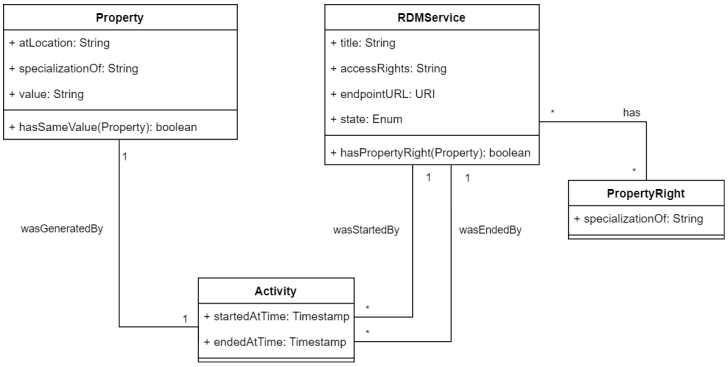


Fig. 6. A class diagram showing the integration service from the logical view.

## 5.3 Logical View

From the perspective of FR4 and FR5 (Section 3.3), the integration service needs to be able to version maDMP properties with their provenance. Additionally, from the perspective of FR1, FR2, FR3, and FR7, the integration service has to maintain information regarding the RDM services and their modification rights. We therefore describe a logical view using the class diagram, see Figure 6.

The class diagram is composed of 4 classes and uses vocabulary from standard The PROV Ontology[11] and Data Catalog Vocabulary[12]. Classes describe data regarding maDMP modifications, data provenance, and RDM services within the institutional ecosystem, as well as information on which maDMP properties have rights. In this way, we meet all functional requirements, but the scheme can be arbitrarily expanded during implementation.

The *Property* class describes all maDMP properties and consists of *atLocation*, which contains the path to the particular object of the maDMP instance, *specializationOf*, which contains information about the type of object and property, and *value*, which stores the maDMP property value. The validity of the values is ensured by the *Activity* class, which consists of two properties, specifically the timestamp when the value became valid and the timestamp when it was changed, i.e. ceased to be valid. Because of FR4 and FR5, the values of the maDMP properties are not deleted but become invalid.

The *Activity* class is linked to the *RDMService* class by two associations that provide tracking of changes by RDM services. The *RDMService* class then describes information about RDM services themselves, including their *title*, *accessRights*, which is also used for their identification, *endpointURL*, which contains information about the remote endpoint to which new maDMPs are sent, and *state*, which describes the state of the RDM service lifecycle.

This RDM service lifecycle consists of 4 states, see Figure 7. After creation, the state becomes *unsynchronized*. It switches to *active* if all instances of the maDMP are successfully transmitted to the RDM service, as it is consistent with the integration service. The RDM service is marked as *unavailable* if it is not accessible during maDMP synchronization. It returns to the unsynchronized state and waits for full synchronization if it then becomes available again. Moreover, it is skipped when sending a new version of the maDMP if the administrators decide that they no longer want it to be part of their ecosystem and change it to the *terminated* state. This state is necessary, since the integration service needs to keep track of the properties' provenance and the RDM service instances are thus unable to remove.

[11] https://www.w3.org/TR/prov-o/
[12] https://www.w3.org/TR/vocab-dcat-2/

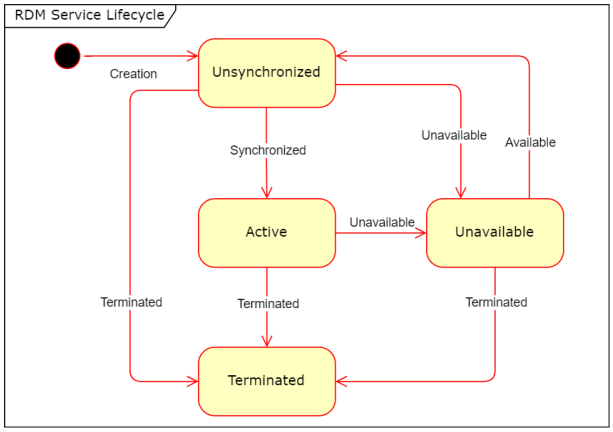


Fig. 7. State diagram displaying the lifecycle of RDM service instances.

The rights of RDM services to individual maDMP properties are captured in their own class *PropertyRight*, which consists only of the already defined property *specializationOf*. Each RDM service instance contains a list of the *specializationOf* values on which it has the right to modify.

### 5.4 Requirements for RDM services

To ensure compatibility between the RDM services and the integration service, we have proposed several requirements and restrictions that must be implemented on their side. The requirements are not described in detail, as each RDM service can have a different implementation, and thus the final design depends mainly on the institutions themselves.

We have identified and modeled two business processes that need to be implemented from the perspective of RDM services. One describes the sequence of steps that occur when a user makes changes within a particular DMP, resulting in new information being sent to the integration service. The second describes a series of steps that must occur when the RDM service receives the customized maDMP from the integration service. Due to their lengths, we have included them in Appendix B.

## 6 Implementation

We implement the **InDMP** application to instantiate the proposed architecture of the integration service. The application uses the Java programming language using the Java Spring Boot framework[13], with the PostgreSQL database system[14] for data storage and the Keycloak application[15] for identity and access management. Most of the logic has been implemented, with the only missing functionality being the retrieval of previous maDMP versions. The source code is open-source, licensed under MIT, and published in the data repository Zenodo[16] and on GitHub[17]. We opt for the Four-Tier Architecture [25], as shown in Figure 8, and divide the logic accordingly into four tiers. Each tier can only interact with the tier above or below it, while the RDM services (clients) interact solely with the presentation tier.

[13]https://spring.io/projects/spring-boot
[14]https://www.postgresql.org/
[15]https://www.keycloak.org/
[16]https://doi.org/10.5281/zenodo.7317353
[17]https://github.com/e11938258/indmp-app

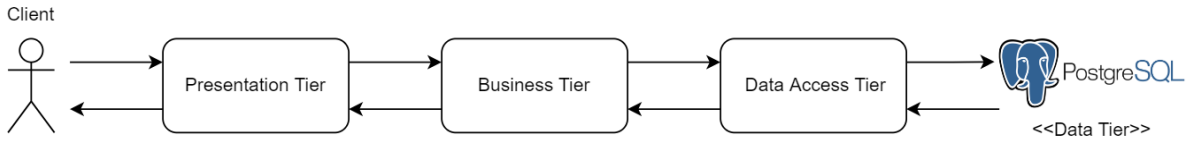


Fig. 8. Applied the Four-Tier Architecture on InDMP application with layer-to-layer communication.

In the InDMP application, information is mapped between the customized maDMP, where the integration service receives new information, and the *Property* class that stores it. The application profile is implemented in the form of Java classes with all restrictions and controlled vocabularies and applies polymorphism and a certain level of abstraction. Specifically, three abstract classes are designed, which include, among other things, the functions that return all non-zero values of each object instance as a list of properties. To retrieve the properties from the received maDMP, these methods are called, one to retrieve properties from the *dmp* object and the second to retrieve properties from its nested objects. The second method then calls itself on all nested objects, resulting in a single list containing all presented relevant properties. During this process, the application checks if the RDM service has modification rights and generates its specialization and location properties. The maDMP is then built in a similar manner. The method for building the *dmp* object is invoked, which retrieves data from the data storage and invokes itself on any nested objects that exist.

To generate the mentioned *specializationOf* and *atLocation* properties, which contain information about the type of object and property and the location within the particular maDMP instance, respectively, two patterns are designed. The first pattern consists of two properties, the *object type* and the *property name* separated by a colon, and is used to construct the *specializationOf*. On the other hand, the *atLocation* property is composed of the identifiers separated by slashes from the *dmp* object up to the object that contains the property, as each object has a binding to this parent, as well as the identifier. For example, if the integration service needs to store the property of the instance of the *distribution* object, the *atLocation* property will consist of three identifiers from the *dmp*, *dataset*, and *distribution* separated by slashes. The only restriction is that instances of the same object type must have a unique identifier, including the *dmp* object.

## 7 Evaluation

We evaluate the implemented InDMP application and the proposed architecture in terms of functional requirements using test cases and level of automation. We execute the application using Java 11.0.13, Keycloak 6.0.1 for authentication and authorisation, and PostgreSQL 10.19 as a database. We generate a collection of test cases using Postman[18] that you can find in the GitHub repository.

### 7.1 Evaluation of Functional Requirements

We design eight functional test cases, see the outline with scope in the Table 4, to simulate a typical interaction between the DMP tool and the data repository to verify the functionality of InDMP. The test cases are designed to discover to what extent our solution corresponds to the defined use cases in Section 3.4 and thus the functional requirements that they model.

The majority of the test cases, which are described in detail in the repository, focus on specific concerns; however, there are also two cases when common behaviour is simulated. We tested erroneous requests to ensure the overall service's coherence as well. Test cases contain three repeating steps, namely: sending a minimum maDMP, a long maDMP, and a maDMP with the dataset. These steps send a request with a maDMP in a body

[18]https://www.postman.com/

Table 4. Summary of functional test cases with their scope depending on the use cases.

| Test case | Test case name | Scope |
|---|---|---|
| FTC1 | Modification of maDMP properties | UC1 |
| FTC2 | Get the maDMP | UC1, UC4 |
| FTC3 | Modification of the maDMP object identifier | UC1, UC2, UC4 |
| FTC4 | Deletion of the maDMP object | UC1, UC3, UC4 |
| FTC5 | Getting provenance information | UC1, UC2, UC5 |
| FTC6 | Verifying RDM service rights | UC1, UC6, UC4 |
| FTC7 | Simulation of production environment 1 | UC1, UC3 |
| FTC8 | Simulation of production environment 2 | UC1, UC2, UC3, UC4 |

Table 5. Summary of level of automation from the perspective of individual use cases.

| UC | UC name | Manual | Semi-Automated | Fully Automated |
|---|---|---|---|---|
| UC1 | Modify maDMP properties | | | ✓ |
| UC2 | Modify the maDMP object identifier | | | ✓ |
| UC3 | Remove the maDMP object | | | ✓ |
| UC4 | Get the maDMP | | | ✓ |
| UC5 | Get provenance information | | | ✓ |
| UC6 | Set RDM service rights | | | ✓ |
| UC7 | Validate and identify maDMP 1 | | | ✓ |
| UC8 | Identify RDM service | | | ✓ |
| UC9 | Synchronize changes with RDM services | | | ✓ |
| UC10 | Synchronize all maDMPs | | | ✓ |



that differs in size - 104B, 1.81KB, and 8.1KB, respectively. In the minimal maDMP, only mandatory properties are sent, while in the long maDMP, a large maDMP with many properties and nested objects is sent. The maDMP with dataset contains the minimal maDMP properties and one very extensively described dataset.

We execute all the functional test cases three times, and at each iteration we receive the correct response code and the corresponding body for all steps. All test cases pass and that the InDMP application meets all the defined use cases and hence functional requirements.

## 7.2 Level of Automation

To assess the level of automation we have achieved in the process of information exchange between RDM services using maDMPs to maintain DMP data up-to-date and consistent, we evaluate the effectiveness of the individual steps from use case scenarios. We identify the scenarios (one scenario for one use case) during the design of the architecture. We use three categories to assess the level of automation that each step can achieve: **manual**, **semi-automated**, and **fully automated**. Table 5 presents the level of automation for each use case.

The aggregation was based on the condition that if a lower level of automation existed in at least one step, we marked the whole use case as such. The results show that all use cases have achieved full automation. Please note that the human work is still needed in modifying maDMP properties itself and managing RDM services within

the integration service, as the operator must specify the scope of modifications. In our assessment, we also did not take into account the human work associated with the maintenance of the hardware, operating system, or third-party applications, such as database systems, etc.

## 8 Conclusion and Outlook

In this paper, we discuss how to employ machine-actionable data management plans within the institutional ecosystem to automate the synchronization of data management information between research data management services, which are used by researchers, as well as other stakeholders throughout the data lifecycle. We present a solution through the design of a conceptual service architecture that is informed by the gathered requirements. This architecture can also handle versioning and tracking of DMP information provenance while limiting the modification rights of RDM services to specific maDMP properties.

As part of the work, we instantiated the integration service into the InDMP application, serving as a proof-of-concept. We created the series of functional test cases to replicate the typical interaction between the DMP tool and the data repository in order to verify application functionality. We evaluated the level of automation of the individual steps of the identified use case scenarios, and the results show that the solution is fully automated, leaving a human to focus solely on managing RDM services within the integration service, as the operator must specify the scope of modifications at a minimum. In order for individual organizations to benefit from the designed service, they still need to customize each individual RDM service to be compatible.

We believe that future work in this area should further focus on decoupling maDMP storage and access functions from individual tools that often provide the presentation layer. For example, DMP tools should help researchers to collect relevant information by providing customized guidance and an intuitive user interface, but they should not maintain their own maDMP store that could become a silo. Instead, DMP tools should persist their information in a service that is shared with other RDM services. In our opinion, this is a necessary step to increase automation and reuse of information, and this paper shows that this is possible.

We aim to use this paper's findings to launch a new Research Data Alliance working group focused on creating a standardized API for DMP platforms. This API will form the foundation of an interoperability framework, enabling new use cases for information exchange in research data management, including DMP platforms, knowledge graphs, FAIR assessment tools, and repositories.



## Acknowledgments

The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme. SBA Research (SBA-K1) is a COMET Centre within the framework of COMET – Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna; COMET is managed by FFG This research is co-funded by the European Union under Horizon Europe OSTrails project 101130187.

## References

[1] Serge Abiteboul and Julia Stoyanovich. 2019. Transparency, Fairness, Data Protection, Neutrality: Data Management Challenges in the Face of New Regulation. *J. Data and Information Quality* 11, 3, Article 15 (jun 2019), 9 pages. https://doi.org/10.1145/3310231

[2] João Cardoso, Leyla Jael Castro, and Tomasz Miksa. 2021. Interconnecting Systems Using Machine-Actionable Data Management Plans–Hackathon Report. *Data Science Journal* 20, 1 (2021).

[3] Paulo A Cauchick-Miguel, Suzana R Moro, Roberto Rivera, and Marlene Amorim. 2020. Data Management Plan in Research: Characteristics and Development. In *Data and Information in Online Environments*. Springer International Publishing, 3–14.

[4] Benjamin Faure, Rob Hooft, Sarah Jones, Georgios Kakaletris, Hanne Moa, Maria Praetzellis, Tom Renner, Brian Riley, Sam Rust, Marek Suchánek, and Diamantis Tziotzios. 2020. Exchange DMPs between DMP Tools using the RDA Common Standard. Zenodo. https://doi.org/10.5281/zenodo.3944308. https://doi.org/10.5281/zenodo.3944308

[5] Sebastian S. Feger, Paweł W. Wozniak, Lars Lischke, and Albrecht Schmidt. 2020. 'Yes, I Comply!': Motivations and Practices around Research Data Management and Reuse across Scientific Fields. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW2, Article 141 (oct 2020),

26 pages. https://doi.org/10.1145/3415212
[6] Sofia Fernandes. 2018. Looking deep at current research information systems: The Information Science perspective in Higher Education. *Qualitative & Quantitative Methods in Libraries* 7, 2 (2018).
[7] Sarah Jones, Robert Pergl, Rob Hooft, Tomasz Miksa, Robert Samors, Judit Ungvari, Rowena I. Davis, and Tina Lee. 2020. Data Management Planning: How Requirements and Solutions are Beginning to Converge. *Data Intelligence* 2, 1-2 (2020), 208–219. https://doi.org/10.1162/dint_a_00043 arXiv:https://doi.org/10.1162/dint_a_00043
[8] Philippe B Kruchten. 1995. The 4+ 1 view model of architecture. *IEEE software* 12, 6 (1995), 42–50.
[9] Christian FJ Lange, Michel RV Chaudron, and Johan Muskens. 2006. In practice: UML software architecture and design description. *IEEE software* 23, 2 (2006), 40–46.
[10] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. 2016. The use and effectiveness of user stories in practice. In *International working conference on requirements engineering: Foundation for software quality*. Springer, 205–222.
[11] Paolo Manghi, Alessia Bardi, Claudio Atzori, Miriam Baglioni, Natalia Manola, Jochen Schirrwagen, Pedro Principe, Michele Artini, Amelie Becker, Michele De Bonis, et al. 2019. The OpenAIRE research graph data model. *Zenodo* (2019).
[12] Tomasz Miksa, Peter Neish, Paul Walk, and Andreas Rauber. 2018. Defining requirements for machine-actionable data management plans.. In *iPRES*.
[13] Tomasz Miksa, Simon Oblasser, and Andreas Rauber. 2021. Automating Research Data Management Using Machine-Actionable Data Management Plans. *ACM Transactions on Management Information Systems (TMIS)* 13, 2 (2021), 1–22.
[14] Tomasz Miksa, Andreas Rauber, Roman Ganguly, and Paolo Budroni. 2017. Information Integration for Machine Actionable Data Management Plans. *International Journal of Digital Curation* 12 (09 2017), 22. https://doi.org/10.2218/ijdc.v12i1.529
[15] Tomasz Miksa, Stephanie Simms, Daniel Mietchen, and Sarah Jones. 2019. Ten principles for machine-actionable data management plans. *PLoS computational biology* 15, 3 (03 2019), e1006750.
[16] Tomasz Miksa, Paul Walk, and Peter Neish. 2020. RDA DMP Common Standard for Machine-actionable Data Management Plans. https://doi.org/10.15497/rda00039
[17] Tomasz Miksa, Paul Walk, Peter Neish, Simon Oblasser, Hollydawn Murray, Tom Renner, Marie-Christine Jacquemot-Perbal, João Cardoso, Trond Kvamme, Maria Praetzellis, Marek Suchánek, Rob Hooft, Benjamin Faure, Hanne Moa, Adil Hasan, and Sarah Jones. 2021. Application Profile for Machine-Actionable Data Management Plans. *Data Science Journal* 20 (10 2021), 32. https://doi.org/10.5334/dsj-2021-032
[18] Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. 2009. From Business Process Models to Process-Oriented Software Systems. *ACM Trans. Softw. Eng. Methodol.* 19, 1, Article 2 (aug 2009), 37 pages. https://doi.org/10.1145/1555392.1555395
[19] Robert Pergl, Rob Hooft, Marek Suchánek, Vojtěch Knaisl, and Jan Slifka. 2019. "Data Stewardship Wizard": A tool bringing together researchers, data stewards, and data experts around data management planning. *Data Science Journal* 18, 1 (2019).
[20] Gustav Pomberger, Walter R. Bischofberger, Dieter Kolb, Wolfgang Pree, and Holger Schlemm. 1991. Prototyping-Oriented Software Development - Concepts and Tools. *Structured Programming* 12, 1 (1991), 43–60.
[21] Stefan Reichmann, Miguel Rey Mazón, Ilire Hasani-Mavriqi, Laura Thaci, and David Eckhard. 2024. D1.1: Plan-Track-Assess Pathways. https://doi.org/10.5281/zenodo.13145788
[22] Andrew Sallans and Martin Donnelly. 2012. DMP Online and DMPTool: Different Strategies Towards a Shared Goal. *International Journal of Digital Curation* 7 (12 2012), 123–129. https://doi.org/10.2218/ijdc.v7i2.235
[23] Stephanie Simms, Sarah Jones, Daniel Mietchen, and Tomasz Miksa. 2017. Machine-actionable data management plans (maDMPs). *Research Ideas and Outcomes* 3 (2017), e13086.
[24] Diana Sisu, Tomasz Miksa, Marie-Christine Jacquemot, Elli Papadopoulou, Maria Praetzellis, and Marek Suchanek. 2024. Elevating data management planning: Interoperability of RDM services through machine- actionability. https://doi.org/10.5281/zenodo.10794257
[25] Ahmad Pahlavan Tafti, Faezeh Rohani, and Mohammad Amin Moghaddasifar. 2012. Towards a scalable G2G framework for customs information system through N-Tier architecture. In *2012 International Conference on E-Learning and E-Technologies in Education (ICEEE)*. IEEE, 175–179.
[26] Ana Trisovic, Philip Durbin, Tania Schlatter, Gustavo Durand, Sonia Barbosa, Danny Brooke, and Mercè Crosas. 2020. Advancing Computational Reproducibility in the Dataverse Data Repository Platform. In *Proceedings of the 3rd International Workshop on Practical Reproducible Evaluation of Computer Systems* (Stockholm, Sweden) *(P-RECS '20)*. Association for Computing Machinery, New York, NY, USA, 15–20. https://doi.org/10.1145/3391800.3398173
[27] Diamadis Tziotzios, Georgios Kakaletris, Georgios Kalabokis, and Elli Papadopoulou. 2020. Import and Export of maDMPs from Argos/OpenDMP. Zenodo. https://doi.org/10.5281/zenodo.3944314. https://doi.org/10.5281/zenodo.3944314
[28] Niklas Zimmer, Norman Smith, Ya'qub Ebrahim, Peter Neish, Joakim Philipson, Brian Riley, and Lyle Winton. 2020. Integrating a Converis CRIS/RIMS with DMPRoadmap and Export/Import maDMP from Figshare. Zenodo. https://doi.org/10.5281/zenodo.3944381. https://doi.org/10.5281/zenodo.3944381

[29] E Zimmerman and Keith G Jeffery. 2004. The Need for a Standardised Current Research Information System (CRIS): A Call to Arms. In *International Conference on Current Research Information Systems*, Vol. 7. 153–160.

# A Process view

## A.1 Business process model for the use case 1

Figure 9 shows the business process model for the task of modifying maDMP properties. It starts when the integration service receives a new version of the maDMP. After identifying the RDM service and the maDMP, the integration service checks all properties from the incoming maDMP against the stored ones, and if they differ and the RDM service has the necessary rights, they are subsequently modified. If the maDMP is not identified, all maDMP properties are stored. At the end of the process, new information is synchronized with other RDM services in the ecosystem.

## A.2 Business process model for the use case 3

The business process model for the situation when the integration service receives a message about the deletion of the maDMP object is shown in Figure 10. It starts with identifying the RDM service and the maDMP. A series of steps is then carried out to verify whether all conditions are met. The important step is that the integration service verifies whether the RDM service has the right to all properties of the object, as well as all nested ones. As long as the conditions are met, all the properties of the particular object and the nested ones are deleted, the *modified* property is changed, and the modifications are synchronized with other RDM services. The process also includes a step in which it is verified whether the object is removable. We do not have any recommendations for this step, perhaps only to prevent the deletion of the *dmp* object, which means that the entire instance of the maDMP is deleted.

## A.3 Business process model for the use case 7

Figure 11 shows the business process model for identifying the maDMP. After verifying whether the maDMP contains mandatory properties, there is an important decision point whether the created and modified dates are the same. If so, the integration service processes it as a new maDMP instance. If not, it processes it as an already existing one and tries to identify it using the *dmp identifier* and/or using the *created* property depending on the configuration.

## A.4 Business process model for the use case 9

Figure 12 depicts the business process model for the synchronization of new information with RDM services. It is done with the help of the customized maDMP, which contains all currently valid property values for the particular instance. It is sent to all RDM services, except the one that initiated the modification. As part of this process, the integration service verifies the availability of all RDM services within the institutional ecosystem and reacts to their unavailability or possible re-availability.

# B RDM Service Requirements

## B.1 Processing New Modifications

Figure 13 shows the business process model for processing new modifications in the RDM service. The process starts with the user, who changes and saves the particular maDMP properties. Depending on the type of modification, the RDM service first modifies the new information in its data storage and then calls one of the three endpoints from the maDMP API. If it is not a change of the object identifier nor the deletion of the object, the RDM service generates the customized maDMP from all its current stored values, which is then part of the message. Each customized maDMP must contain at least the mandatory values. It means that each RDM

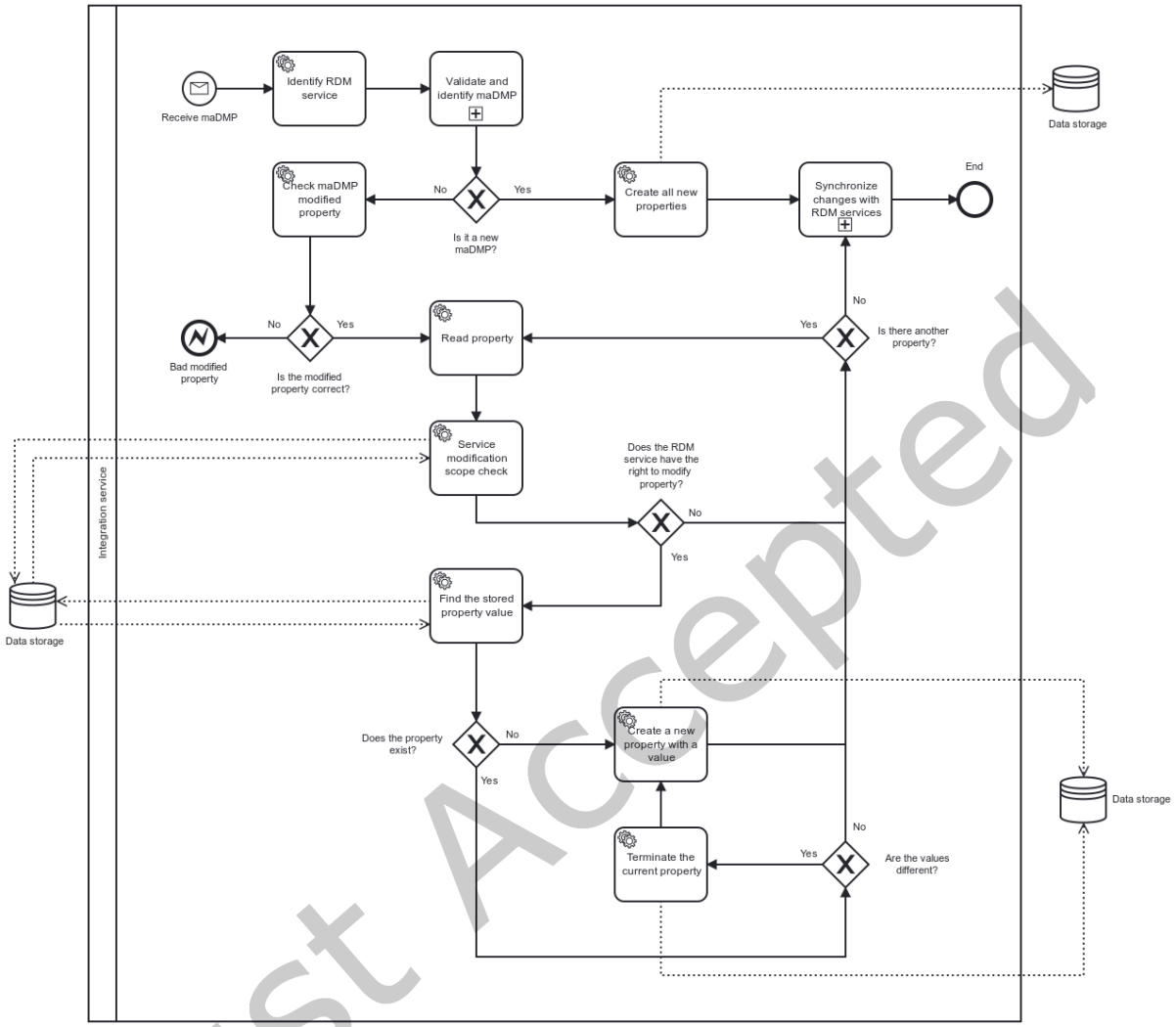


Fig. 9. Business process model for the use case 1.



service, which wants to be able to send new modifications, must have these values stored as well as implement the customized maDMP.

## B.2 Processing of Incoming MaDMPs

Figure 14 depicts the business process model for processing of incoming maDMPs in the RDM service. The process starts by receiving the message with a new version of the customized maDMP. The RDM service then validates and identifies the incoming maDMP. If it is a new maDMP, it saves mandatory and relevant information from it and checks whether it possesses other information that is not in the current maDMP. If it has, it generates the customized maDMP from all stored values and sends it back to the integration service.

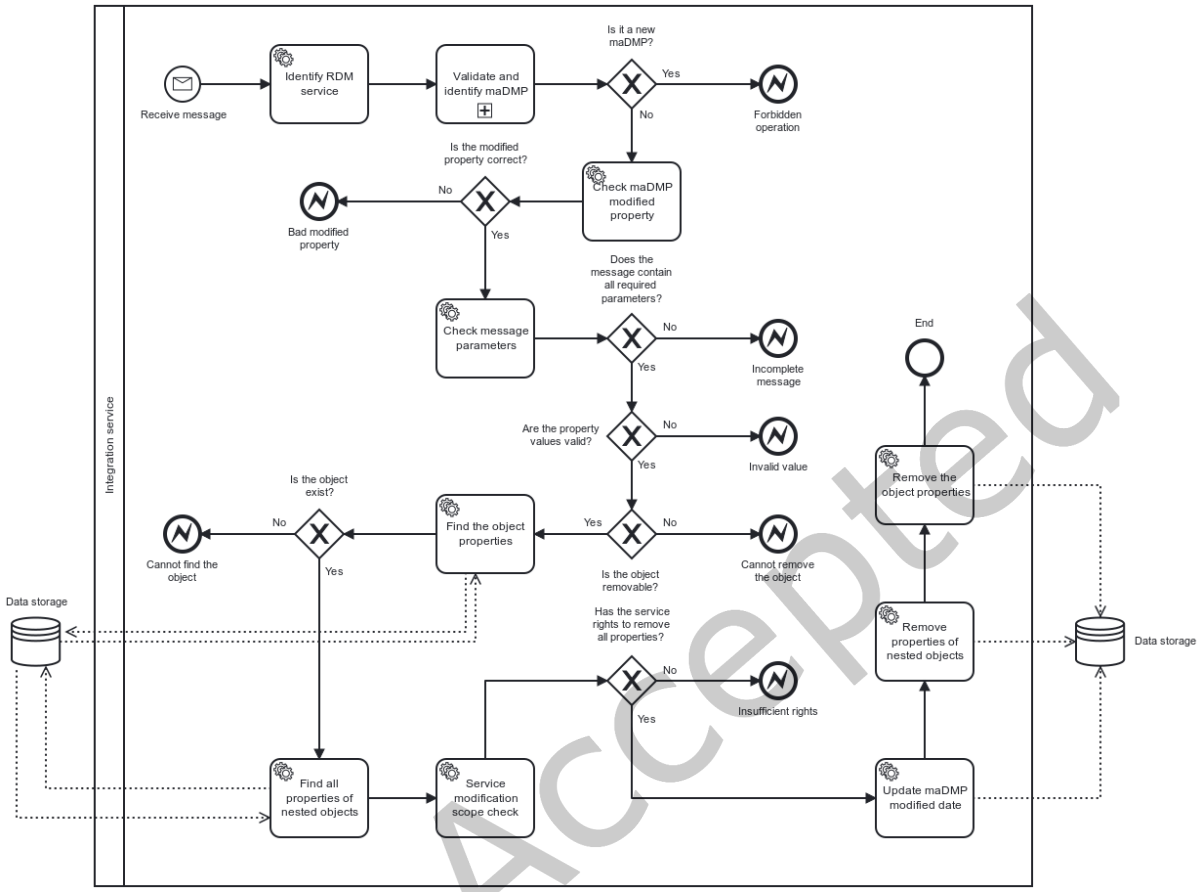


Fig. 10. Business process model for the use case 3.

If the maDMP was identified, the RDM service extracts all relevant information from it and synchronizes DMP properties with the stored ones. If it cannot identify all stored objects during synchronization, it requests the integration service to provide provenance information about the specific type of objects, from which it can determine whether the object was deleted or its identifier was changed. At the end, it checks whether the maDMP contains new relevant objects to store. Each RDM service thus has to store the identifiers of all the objects from which it draws maDMP properties. The business process model for the use case 7 defined in Appendix A is recommended for identifying maDMPs.

Received 9 February 2024; revised 8 November 2024; accepted 1 January 2025

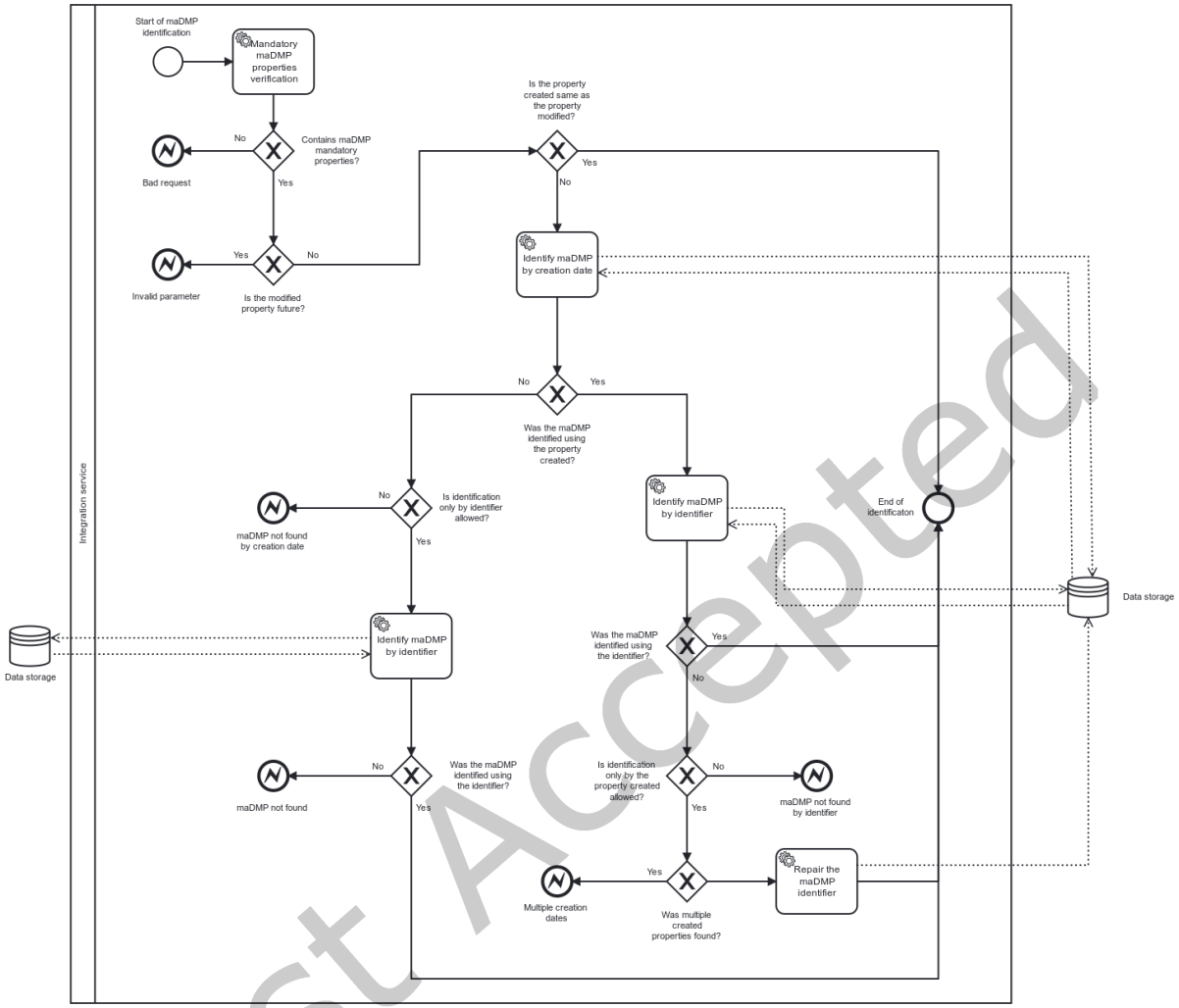


Fig. 11. Business process model for the use case 7.

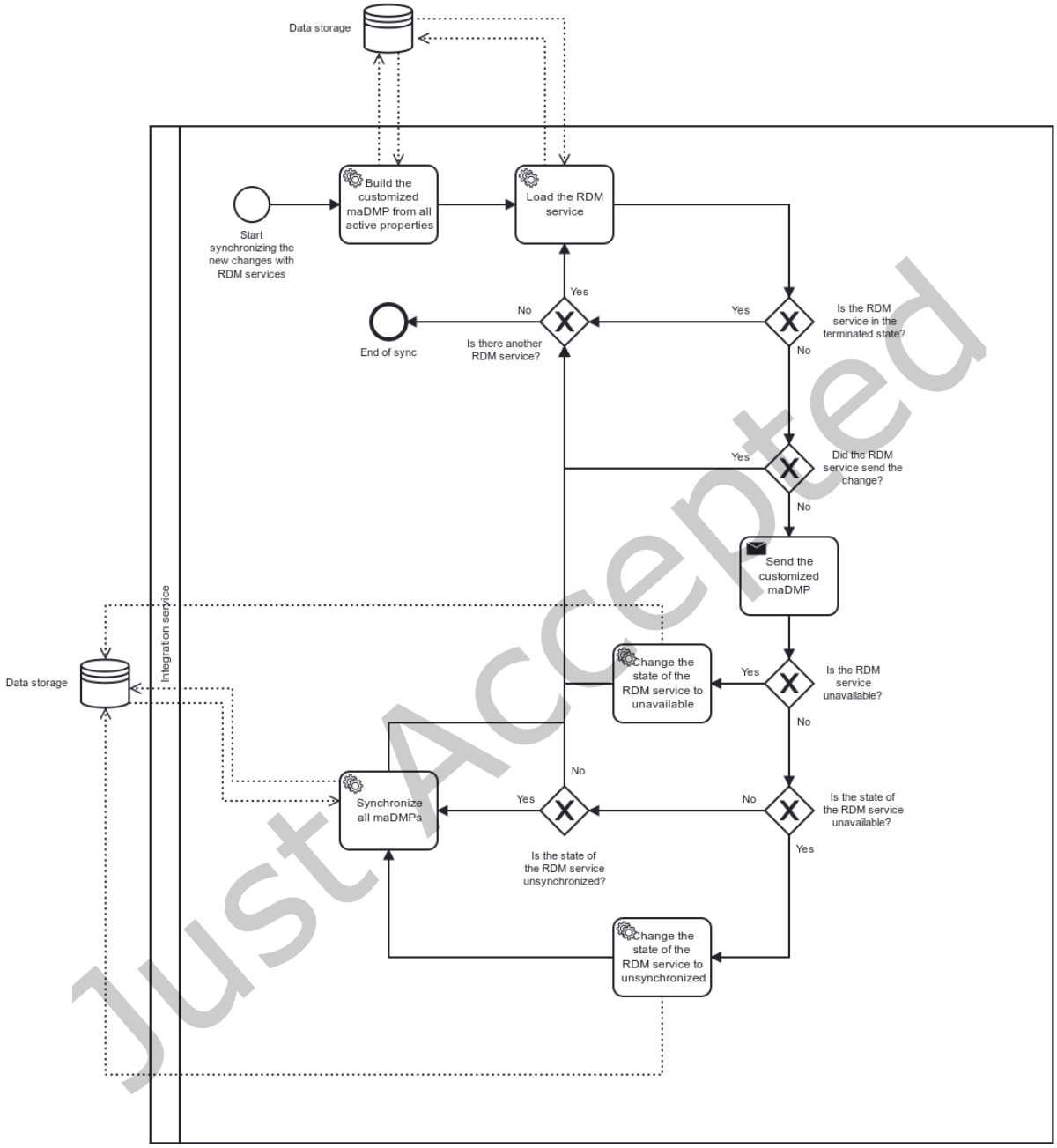




Fig. 12. Business process model for the use case 9.

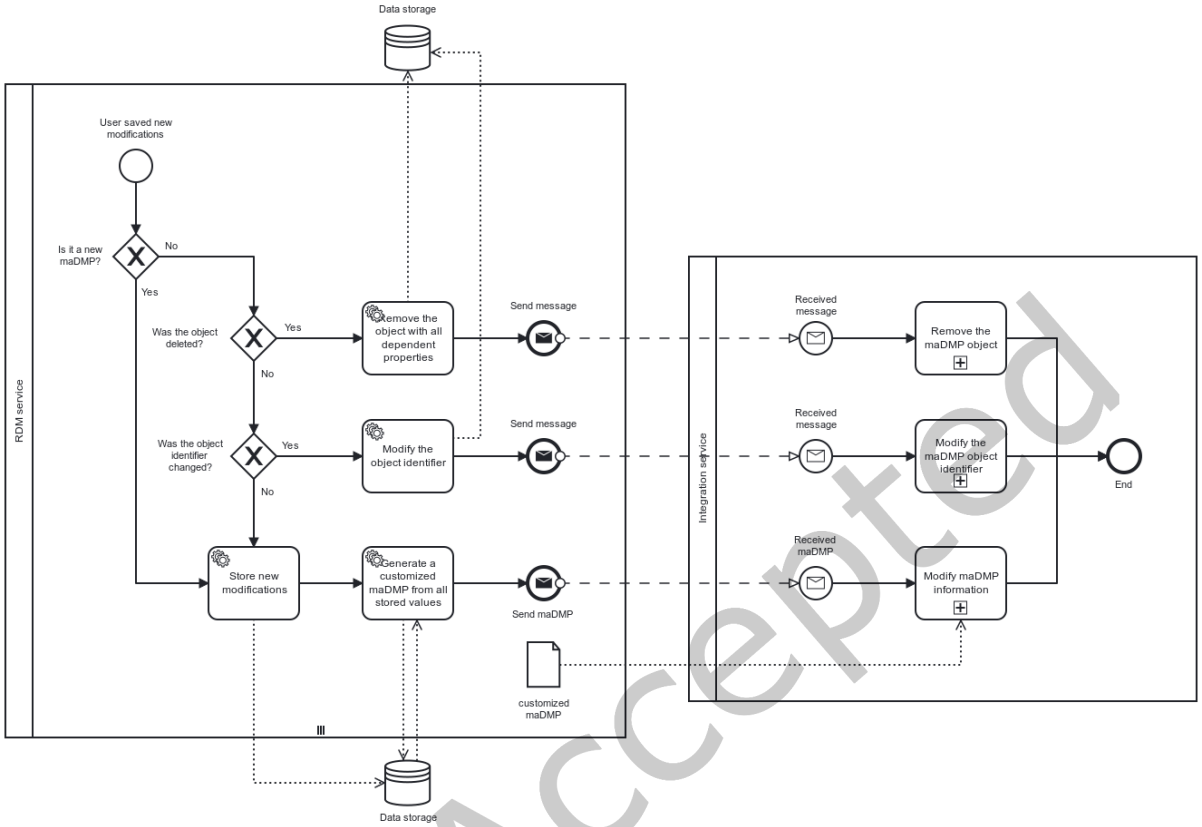


Fig. 13. Business process model illustrating the behavior of the RDM service upon a new change from the user.

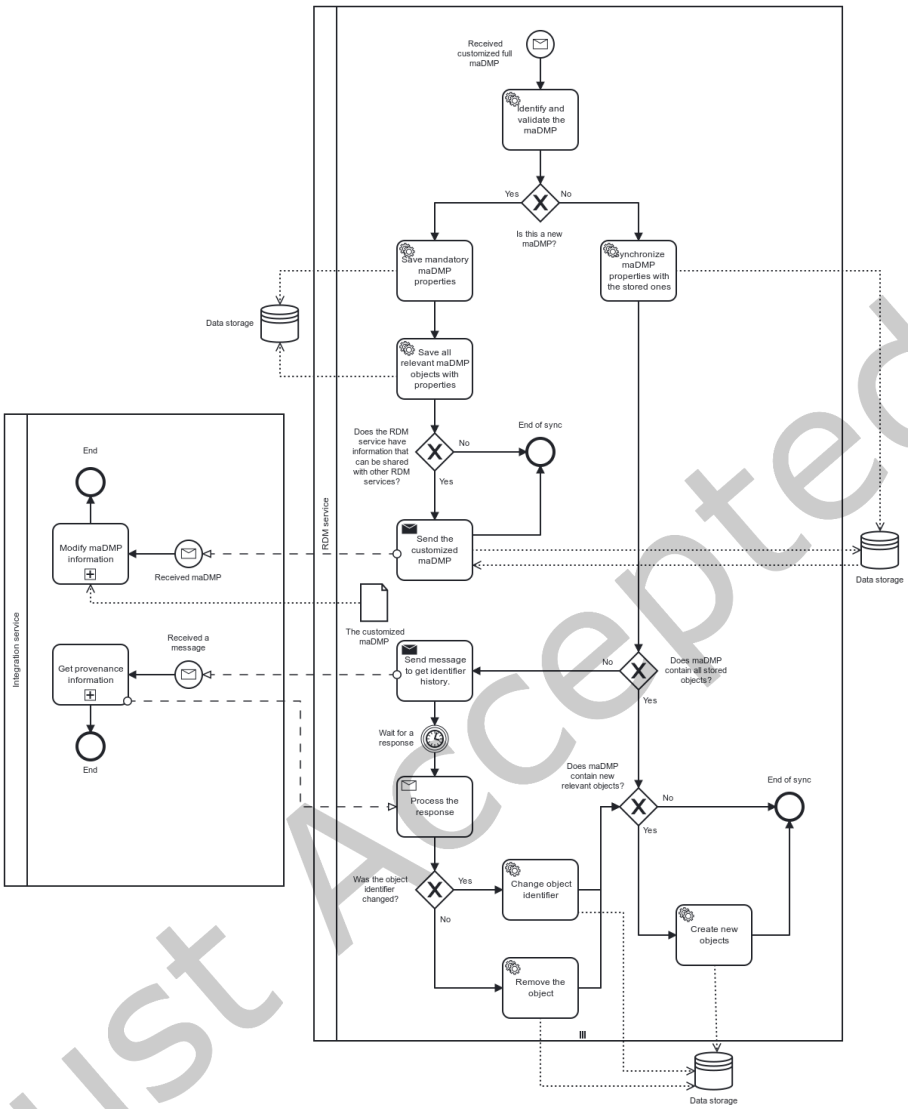


Fig. 14. Business process model illustrating the behavior of the RDM service when receiving a new maDMP.