Published in partnership with the Shanghai Institute of Ceramics of the Chinese Academy of Sciences

https://doi.org/10.1038/s41524-025-01618-1

An extensible open-source solution for research digitalisation in materials science

Victor Dudarev, Lars Banko & Alfred Ludwig

Information technology and data science development stimulate transformation in many fields of scientific knowledge. In recent years, a large number of specialised systems for information and knowledge management have been created in materials science. However, the development and deployment of open adaptive systems for research support in materials science based on the acquisition, storage, and processing of different types of information remains unsolved. We propose *MatInf*—an extensible, open-source solution for research digitalisation in materials science based on an adaptive, flexible information management system for heterogeneous data sources. *MatInf* can be easily adapted to any materials science laboratory and is especially useful for collaborative projects between several labs. As an example, we demonstrate its application in high-throughput experimentation.

The further development of modern science and technology relies strongly on the collection, curation, processing and use of data and relevant metadata: data-driven science is now more relevant than ever[1]. However, in materials science, despite the growing number of specialised information systems on the properties of materials, there is a lack of flexible, independent open-source systems capable of successfully solving the problem of storing and retrieving information about materials and their properties and easily adjusting to support emerging data formats. Successful attempts to develop such fully open-source extendable systems—including both simulated and experimental data—are not known. This might be related to the complexity of the domain and the unavoidable emergence of specific data formats or information structures, which are difficult, if not impossible, to take into account within standalone solutions[2]. Data management is challenging since the system should be flexible to process heterogeneous data formats from proprietary binary data (from measurement devices, e.g., raw files from an X-ray diffraction system) to well-structured and easily parsable XML/JSON/CSV data.

We present *MatInf*—a highly extensible and customisable platform for materials data management. For the development and implementation, experience from materials science and software engineering were combined. The implementation was supported by extensive usability testing and design reviews of existing data management systems, system analysis and relational data modelling together with UX/UI design and Web development. An analysis of existing Electronic Lab Notebook (ELN) platforms showed that current open-source ELNs: e.g., eLabFTW, openBIS (and other software which are not ELNs but are sometimes used in this role: OpenWetWare, JupyterHub) are a good starting point for managing scientific data, however, they lack user-defined data types support and customisability and hence are not really able to tackle complex data types properly and perform support and effective search for materials data.

A further motivation for our development was gained through recognising that existing research data management systems (RDMS) offered e.g. within NFDI initiatives, such as NOMAD (FAIRmat), Kadi4mat (NFDI4Ing), and Chemotion (NFDI4Chem)[3–5], do currently not fully address the needs of experimental high-throughput materials exploration workflows[6] within a single information system. While NOMAD, developed by FAIRmat, is one of the most advanced systems in this context, it does not natively support the specific workflows and experimental data types "out of the box", which are required e.g., for combinatorial synthesis of thin-film materials libraries and their high-throughput characterisation. Implementing new experimental data types in NOMAD often necessitates significant custom development, including the creation of plugins that can function locally but lack integration with the central NOMAD repository. Additionally, our workflow relies on close integration and interaction between objects, such as enriching existing objects with new data during the addition of new ones—capabilities that are currently unavailable in other systems. These challenges, combined with the need to manage heterogeneous

Materials Discovery and Interfaces, Institute for Materials, Ruhr University Bochum, Bochum, Germany. e-mail: victor.dudarev@rub.de; lars.banko@rub.de; alfred.ludwig@rub.de

data, and bridge theoretical and experimental data outcomes, emphasize the necessity of our system, *MatInf*.

This article aims at developing a high-level metadata model in materials science, able to be a common ground for materials data, and its implementation in the framework of an open-source system under continuous further development, based on a flexible system of extensible data types (https://gitlab.ruhr-uni-bochum.de/vic/infproject). The current results of the system development and use for representing heterogeneous data on materials properties are demonstrated, and the extension of the system types based on external Web services is briefly discussed. So far, the system is being successfully used in several tenants by geographically distributed collaborating research teams dealing with various tasks (computational and experimental materials science and chemistry) and handling different document formats.

Results

Defining requirements

Based on our experience in software systems and the materials science domain we formulated requirements for an RDMS capable of materials science data support and specifically to be flexible and powerful enough to tackle changing and emerging data formats used in high-throughput materials exploration tasks. The system could also be used in other or less demanding research environments and by being able to handle high-throughput workflows it also lays a basis for future autonomous experimentation. The source code is open and the following functional requirements were put forward for the development of the flexible system.

(i) Account support for individual systems (tenants—independent system instances with their settings/users/data) needs to be implemented. If a new system instance needs to be added, it must be implemented fully independently (separate database, no dependency on proprietary solutions) or be able to share data types and users with other tenants[7].

(ii) A user registration (including e-mail verification) and an authorisation subsystem need to be developed, including the use of external OpenID Connect-compliant authorisation centres[8]. The system also requires an administrative subsystem for user management, which can be accessed by users with system administrator permissions[9].

For simplicity, the system introduces three predefined user roles.

- *Administrator*: authorised to take any action in the system, including deleting objects and managing users.
- *Power user*: authorised to add and edit objects they have created.
- *User*: has read access rights to protected objects.

If a user does not belong to any of the roles (by default immediately after registration), only access to publicly available information is possible. All information objects in the system have one of three access levels (which is set when the object is created), which determines the visibility of the object for authorised users (the present access control system offers minimal implementation and performance overheads for security checks; it is possible that the system will be revised in the future in order to support access control lists (ACLs), allowing a more granular level of access control):

- *Public*—available to everyone, including unauthorised users.
- *Protected*—available only to authorised users who are assigned to at least the *User* role.
- *ProtectedNDA*—available only to authorised users who are assigned to at least the *User* role and have *NDA* (Non-Disclosure Agreement) claim (or in an *Administrator* role).
- *Private*—only available to the current user (author).

It should be allowed to load and store different types of objects in a structured way using a user-defined hierarchical classifier. In this way, users can create tree-like rubrics as containers for objects and thus structure the data. It is also possible to establish directional associations between objects, allowing the creation of graphs of interrelated objects.

As the domain of knowledge is defined by materials science, basic type support is required to represent materials objects: material systems, materials and their modifications. The following definitions are introduced:

- *material system*—a set of chemical elements that built up a material, denoted as S. Each chemical element is denoted by a unique identifier e_i, then the chemical system S is a set $\{e_1, e_2, \ldots, e_n\}$, in other words, the term "material system" expresses a qualitative composition of a material (e.g., Ni-Ti, Fe-Co-O).
- *material*—a quantified chemical composition, i.e., a set of chemical elements with their general content (if solid solution is considered) or a specific content (in the case of single-phase compound, stoichiometry) that built up a material. Let's denote n_i to represent the quantity or coefficient associated with a particular element e_i, then material, designated C, is a set of pairs: $C = \{(e_1, n_1), (e_2, n_2), \ldots, (e_n, n_n)\}$ (e.g. Co_3O_4, Li_3NbO_4).
- *modification*—E.g., a particular phase, e.g. crystalline or amorphous structure, form (e.g., thin film, powder, bulk) of a material could be expressed as a string.

The RDMS supports all the above-mentioned materials objects. Since different types of objects can have an arbitrary set of properties, a mechanism is provided to associate a set of properties of different types with an object, defined at the object type level. A template feature has been developed that allows implementing any table structure based on the properties that contain the data/metadata according to the project knowledge/agreements. Therefore, it is recommended to set up a template for each user-defined type to enable and maintain the object's data integrity. In addition, the import and export of tabular data associated with the object using Excel and the support of Excel templates for tabular data is implemented.

A search interface is developed that supports searches both by built-in object types, including chemical objects (material system, material, modification) and by object properties based on user privileges.

The core of a flexible system is the development of an extensible object type system that not only supports the creation of new object types with their own set of properties based on predefined data types but also allows the support of new data types by using external, type-supporting services via API. This should perform three main functions: validating data, providing data for import (material systems, materials, properties) into the system to organise search and direct access to data, and data visualisation.

The information system, hereafter denoted as *MatInf*, should provide an API to access data in the system for use in external systems. In addition, visualisation of standard tabular data, and construction of point graphs of dependencies on user-selected axes should be built-in. Other functionality, including the construction of specialised reports, charts, and graphs is subject to expansion of the system using external services and APIs.

Architecture outline of the extensible object structure

In the context of the mentioned requirements, we consider the main implementation aspects of the system. Architecturally it is a modular Web-solution based on ASP.Net Core technology and a relational SQL Server database backend. This combination proved its high performance and reliability. The usage of the Model-View-Controller design pattern together with high modularity makes the resulting system flexible (see Fig. 1). Abstraction layers, introduced on the top of data and APIs, enable effective data manipulation.

The system is based on the storage of strongly typed objects in a relational database, where object characteristics (or properties) can be stored in multiple tables. It is important to note that the object definition is introduced by analogy with object-oriented programming, so it is important to define the root object of the hierarchy from which all characteristics are inherited.

Root class type

The root class is *ObjectInfo*, which encapsulates properties inherent to all objects, such as a unique object identifier (*ObjectId*); tenant affiliation; date and time of object state creation and modification, and identifier of the user who performed the action; object type identifier (*TypeId*); identifier of object affiliation with the rubric (*RubricId*); sort code (*SortCode* is set if it is necessary to sort objects when outputting in a special way apart from sorting by name, so the *Order By* clause in SQL is "*SortCode, ObjectName*"); object access level (public/protected/protectedNDA/private); optional external object identifier (*ExternalId*—may be necessary for service purposes, e.g., when synchronising the database state with an external resource); object name (mandatory field, must be unique within (*TenantId, TypeId*) pair to prevent wrong naming); object name for building the URL; optional path to the object data file (*ObjectFilePath*); SHA-256 hash of the data file (*ObjectFileHash*, to trace uniqueness of data in the system by building a unique index by the pair (*TenantId*, *ObjectFileHash*) for filled hashes); optional object description.

All derived classes of objects in the system, such as materials or literary references, are inherited from the *ObjectInfo* type and, in addition to the above-mentioned properties stored in the *ObjectInfo* table, store their state in other system tables (for natively supported types) and extended property tables to store integer, real and string values of object properties (for types supported over native types due to system extensibility).

Materials classes as derived types

Each object belongs to a particular object type, which is described in the relational table *TypeInfo* (referenced by the *TypeId* attribute). In this context, object types are extensible—a new entry for an object type must be added by referring with the *TableName* attribute to one of the built-in types (*RubricInfo*—to store hierarchies such as projects/subprojects or organisational structure) for objects: *ObjectInfo*, *Sample* (material system), *Composition* (material), *Reference* (literary reference).

As shown in Fig. 2 top-level chemical entities like material system and material are first-class supported, since according to their description in the section "Defining requirements" they are reflected in dedicated tables (*Sample* and *Composition*). This allows the system to interact with corresponding materials objects faster since data structures are optimised at a database level.

Similarly, a *Reference* table for literature references support was implemented and connected to *ObjectInfo* by 1-to-1 relation (like *Sample* table, but for the sake of simplicity it is not shown in Fig. 2). Note that each introduced object type may be supported by a typed set of extended properties.

Class types extension

Each object can be described by a set of typed properties stored in property value type-dependent tables, as shown in Fig. 3: integer values are stored in *PropertyInt* table, real values in *PropertyFloat* table, string values in *PropertyString* table. Property names are defined by the *PropertyName* attributes, all values are stored in the *Value* attributes of the corresponding *Property** tables. The build-in separator-property mechanism together with the opportunity to build complex multi-level property names allows to represent hierarchically arranged properties. This feature is useful when the object has a huge number of parameters that should be hierarchically structured for usability and user convenience. The *Row* attribute allows, if necessary, by specifying the row number, to form a virtual table from the values of the extended properties. This ensures the efficient storage of sparse table data. Despite the extreme representative power of this mechanism, it should be used wisely. If the number of objects using it exceeds 100k and the data in this table is not sparse, we recommend creating *ObjectInfo*-derived table structures, as it was done with *Sample* and *Composition* in Fig. 2.

Establishing a dependency graph

It is often necessary to establish links between objects in the RDMS, e.g., a materials library (consistent collection of materials samples, manufactured under the same conditions[10]) needs to be linked to its measurement results (e.g., measurements of different physico-chemical properties at all defined measurement areas). Thus, importing a compositional analysis document of a materials library not only results in a document of measurement results linked to the library object but also creates linked objects characterising the quantitative composition of the material in each measurement area (region of a materials library). The directed linking of objects to each other and the assignment of appropriate semantics to links according to the objects' types enables the construction of a graph representation of objects in the materials science domain in question (performed via the *ObjectLinkObject* table).

API for flexible user-defined types support

Given that, each object can contain a link to a data file (the data obligation for the object type is set in the *TypeInfo.FileRequired* settings), the crucial concerns are data validation, importability into the system, and visualisation. Considering the heterogeneity of data in materials science (tabular data in different formats, images, binary files from measurement devices) exchanged by scientists during research, it is impossible to develop a

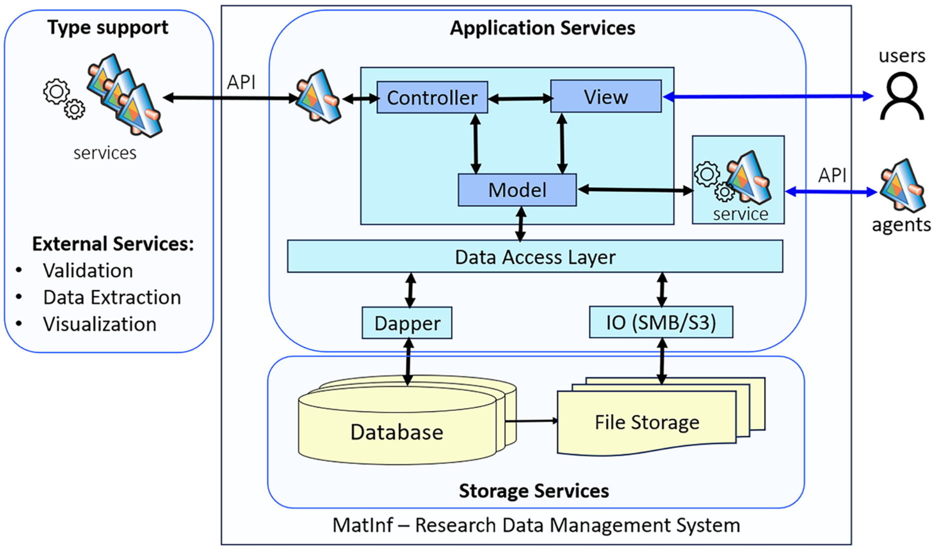


Fig. 1 | Schematic of the extensible *MatInf* architecture supporting user-defined types via external services (left part) and flexible data access through API (right part). Web-application is developed using a classical three-tier architecture consisting of clients, application server and data storage subsystem. Clients (located in the figure on the right) can be both end users using browsers and external software environments/agents interacting with the system via REST API. The central part of the system is the application server running the Web services and Web-application based on the model-view-controller (MVC) architectural pattern. System data models interact with the data access layer, which consists of two components: the relational data handling module (built on the basis of Dapper ORM) and the input-output subsystem, responsible for storing documents by means of both local file systems and server message block (SMB) or S3 (simple storage service) systems. The basis of the extensible type system inherent in *MatInf* is the opportunity to delegate to external services, implemented in accordance with the developed specification, the tasks of document validation, data extraction and visualisation (left part).

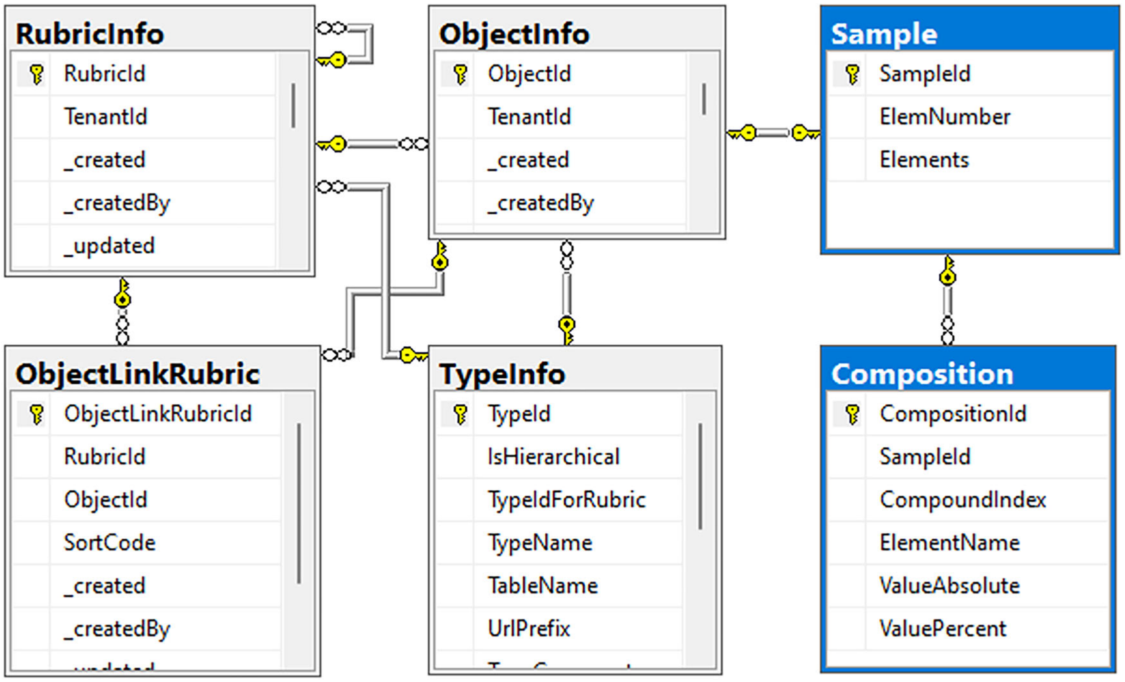


Fig. 2 | *Sample* and *Composition* tables for materials classes supported by *ObjectInfo* extension on a database level. Extensible system of system object types is based on the *TypeInfo* table, each row of which stores information on a type. Each type can be either hierarchical (*IsHierarchical* = 1), in this case, type objects are stored in the *RubricInfo* table and represent a tree structure, for example, a classifier), or list (*IsHierarchical* = 0), then type objects are stored in the *ObjectInfo* table and represent a type-specific extensible set of properties. For example, there is built-in support for material science entities, such as chemical systems and compounds, whose data are stored in the *Sample* and *Composition* tables in addition to the base *ObjectInfo* table. Each object in the list must belong to one of the classifier nodes to provide convenient user navigation (the *ObjectInfo.RubricId* field is associated with *RubricInfo.RubricId*). Additionally, using the *ObjectLinkRubric* table, each object of the list type can be linked to other classifier nodes or participate in classifiers of other types, which allows flexible management of different object categorisation.

common data schema that would cover *all* possible formats. Therefore, the decision has been made to delegate these tasks to third-party software components, accessible either through late binding via an interface link or using external web services accessible via HTTPS through a formalised API. Assigning the task of working with data types unknown at the time of system design to third-party services with a formalised response scheme allows files of different formats to be processed with equal success since external software components are responsible for analysing them and returning JSON documents in a standardised format.

Data validation

In addition to the mandatory validation for the uniqueness of stored files for each object type (at *TypeInfo* table level), one can specify an additional validation method by modifying *TypeInfo.ValidationSchema* value. Options are either a built-in validator supporting *IFileValidator* interface, which instance is created dynamically using Reflection (using "type:" prefix), or an external validator available via REST API (using "https:" prefix). In the second case, the document for validation is sent to the address specified in the settings in the POST request body, and the service returns serialised in a JSON object of *TypeValidatorResult* type with validation results described by three properties:

- *Code—int* type, error code: 0—validation succeeds; ! = 0—validation fails.
- *Message—string* type, error message if validation fails (*null* if validation succeeds).
- *Warning—string* type, warning on successful validation (otherwise *null*). Message that may cause an error in the future (in case of rules strictification).

So, if the document validation is successful the response returned is {"Code": 0, "Message": null, "Warning": null}. If the validation failed, {"Code": 500, "Message": "Error description", "Warning": null}. The service specification is available in OpenAPI format, which simplifies the development of similar services to support new object types.

Data import

For data search in *MatInf* and indexing of uploaded documents, special support for data import schema is introduced at the type level, supporting the addition of chemical objects as well as extended properties when a document is uploaded. This support, architecturally similar to validation, can be implemented either at the embedded object level with the help of predefined programming interfaces or at the level of external web services accessible via REST APIs. A complete specification for the import data formats is not possible here, however, the formats from the DTO object specifications are available in the source code of the system (see *TypeValidationLibrary* project).

Data visualisation

The RDMS has built-in support for visualising graphics in standard formats and tabular data, with the functionality to sort them and build plots on selected axes. The visualisation of materials library data is implemented in *MatInf* (Fig. 4). A more flexible visualisation of type data can be configured using an external web application which is accessible via the HTTP(S) protocol.

An external rendering address can be specified for the object type, capable of accepting a POST request from a system with data in the body of the request. Having received the data, the external system independently renders the document and is responsible for interaction with the user.

Data search

The data search is materials-oriented and offers flexible searching options to find the required objects. The flexible type system plays a major role in this,

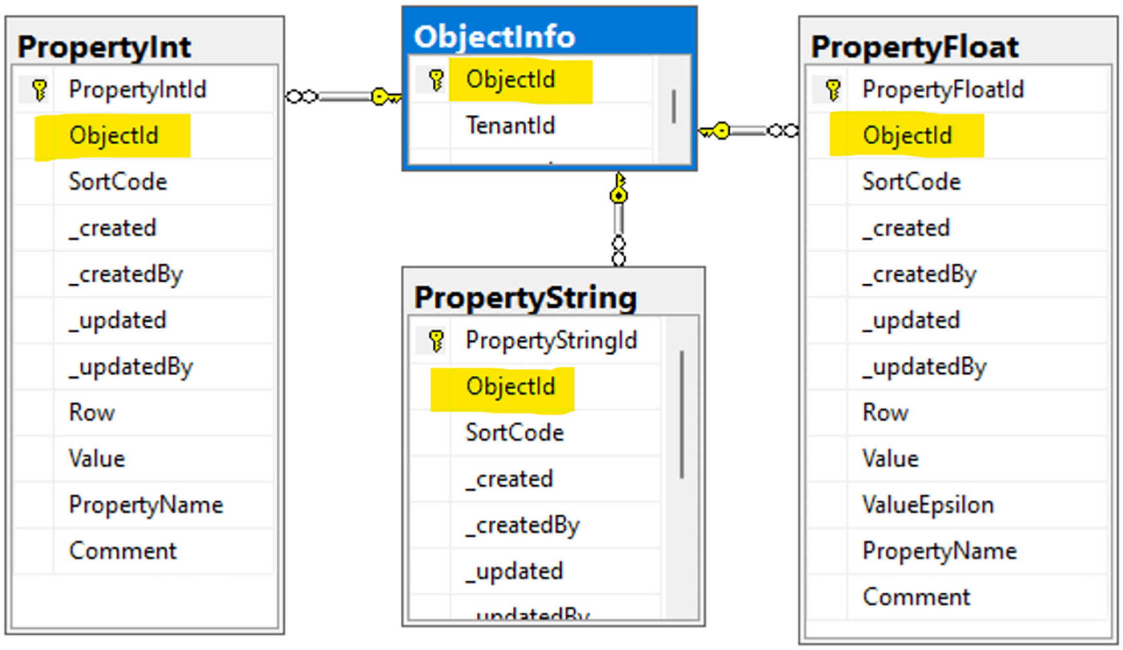
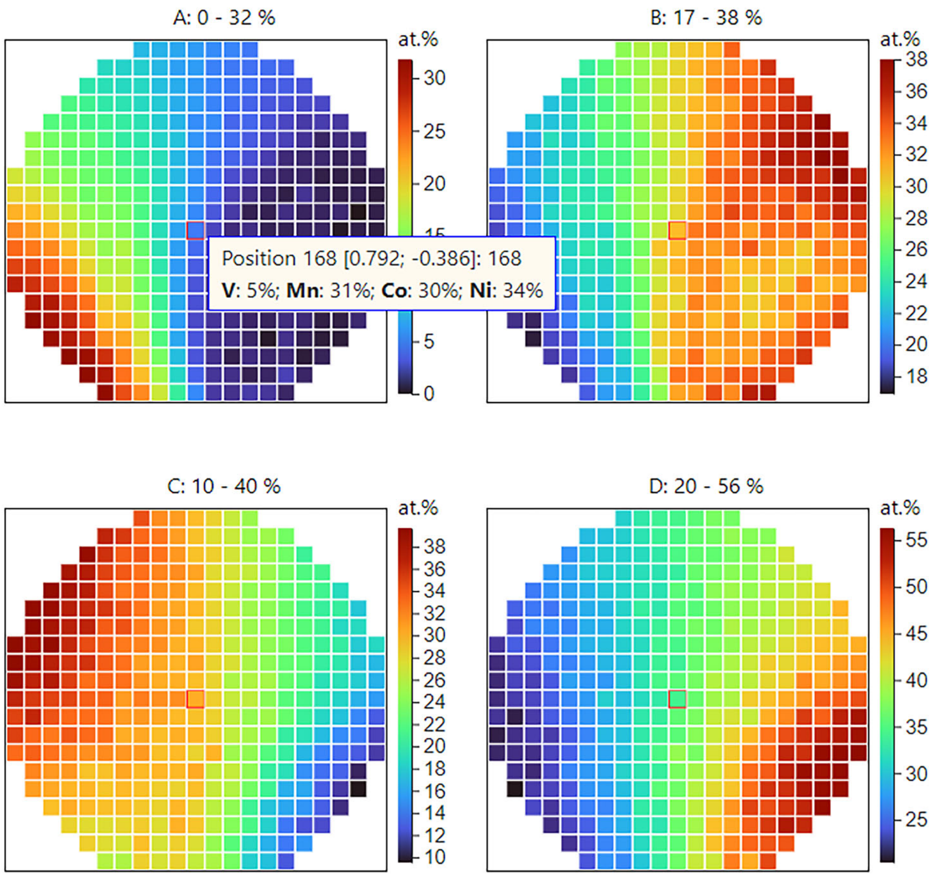
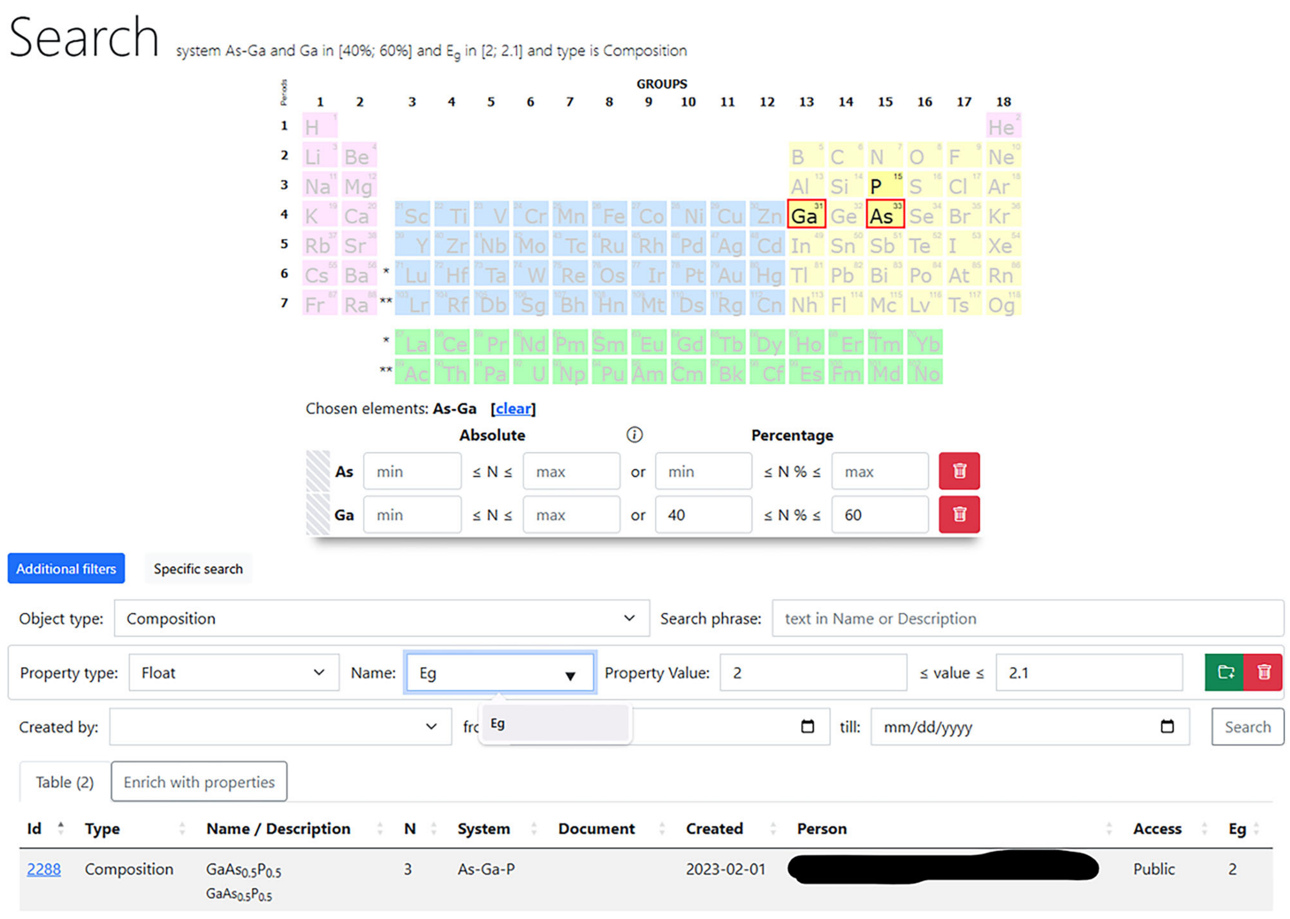


Fig. 3 | Extended object properties, stored in a strongly typed manner, enabling reach search functionality. Each object of the list type in the system has an entry in the *ObjectInfo* table. This table is connected with *Property** tables for storing typed properties of objects (*PropertyInt* for integers, *PropertyFloat* for real values and *PropertyString* and *PropertyBigString* for strings). The set of properties for an object is a union of the data from the four tables above. Each property is associated with the object (via the *ObjectId* value) and has a name (*PropertyName*) and a value (*Value*—the type is determined depending on the table), and a row number in the table (*Row*—used only if a tabular representation of the data is required; when *Row* = NULL, the property is a key-value pair). To specify the order, all properties have a sort code (*SortCode*) that allows controlling the sequence of property output. Unique indices on attributes (*ObjectId*, *Row*, *PropertyName*) allow to maintain uniqueness of named property values for objects.

Fig. 4 | Example of a materials library automatic visualisation by *MatInf* showing colour-coded compositional information of a quaternary system. The system has built-in support for material library properties visualisation in the form of a colour (temperature) coding scheme of property values on the surface of the materials library, where blue colour corresponds to the minimum values of the measured property, and dark red—to the maximum. In this example, information about the change in the quantitative composition of compounds in different measurement areas of a materials library consisting of four components is visualised. Colour coding allows a visual assessment of how property values change as one moves from one measurement area to another by observing the colour gradient. Displaying multiple properties in parallel allows us to explore the relationships between different properties.




Search system As-Ga and Ga in [40%; 60%] and E_g in [2; 2.1] and type is Composition

Chosen elements: As-Ga [clear]

	Absolute				Percentage		
As	min	≤ N ≤	max	or	min	≤ N % ≤	max
Ga	min	≤ N ≤	max	or	40	≤ N % ≤	60

Object type: Composition Search phrase: text in Name or Description

Property type: Float Name: Eg Property Value: 2 ≤ value ≤ 2.1

Created by: from: till: mm/dd/yyyy

Id	Type	Name / Description	N	System	Document	Created	Person	Access	Eg
2288	Composition	$GaAs_{0.5}P_{0.5}$ $GaAs_{0.5}P_{0.5}$	3	As-Ga-P		2023-02-01	[redacted]	Public	2

Fig. 5 | Example of "Search Results" for "Composition" of samples containing Ga (in the range 40–60%) and As (any content) with "Bandgap" in the range 2–2.1 eV. Built-in search form integrated with the periodic system allows to find information about objects contained in the system by many different criteria: from qualitative and quantitative composition of material to values of specific properties (for numeric types search by interval is allowed, for string types by substring). Metadata related to the date of object creation, authors and other related information can also be used to find the required data.

which makes a decisive contribution to the information content of the database when importing heterogeneous documents. The user interface of the search subsystem is presented in Fig. 5. As basic fields the search form includes such items as: type of object, phrase from the name or description, user-creator, date of creation and service identifiers. The search form is dynamically adjustable allowing the user to add properties to search in an underlying type-aware way: numbers could be searched within a specified range and strings using substring search. Moreover, search is facilitated by automatic suggestions (autocompletion) for parameter names of user-selected types by looking up database content. One of the crucial advantages of the current software system is the ability to search for material entities using object properties together with implemented persistent search URLs that can be shared to reference search results. It should be noted that the search is carried out taking into account the current security context (depending on the authorised user and his membership to roles) and the level of availability of objects in the system.

Discussion

The open-source *MatInf* software is under continuous development and belongs to the class of Research Data Management Systems for materials science, since it provides support for materials science object types. The extensibility of the system by adding and supporting validation, importing and mapping data from added types through the use of APIs gives the system considerable versatility, allowing flexible work with heterogeneous data types. Support of multiple tenants allows not only the isolation of system instances but also fast addition of new instances, which gives the opportunity to provide the developed system as SaaS (Software as a Service) solution for different workgroups. External interface and object type system options allow for flexible customisation. Further development steps are aimed at enhancing the customisation of individual tenants and implementing an API for direct input of data into the system e.g., from measurement devices or custom data sources, which will enable automated research data collection and further analysis, including the use of third-party software systems communicating via API. We believe that the documentation (https://inf.mdi.ruhr-uni-bochum.de/home/doc), as well as source code templates for API services to support user-defined types, will foster a community around *MatInf* interested in developing the system and connecting new data formats. This will contribute to advancing digitalisation in materials science and improving the efficiency of research data usage. *MatInf* is open to sharing functionality and ideas for a related RDMS, e.g., from NFDI consortia and other interested parties.

Methods

Set theory and tuples

Top-level modelling of the basic objects of materials science has been done using set theory and tuples. These mathematical constructs enable naturally representing material systems as a set of chemical elements and material as a set of ordered tuples, enriching every element with a number representing its content. On the bottom levels description could be extended by either extending tuples or introducing new sets.

Relational data model

Having defined basic chemical entities with set theory it was a straightforward decision to rely on relational database theory to develop an appropriate data model to flexibly represent objects of different types. In the article, we provided only a limited number of examples showing modelling of material systems and material (Fig. 2) together with properties (Fig. 3) enabling lightweight key-value descriptions and arbitrary table representations. The complete database structure can be found in the documentation section of any tenant, for example, https://inf.mdi.ruhr-uni-bochum.de/home/doc.

Service-oriented architecture

Service-oriented architecture is the heart of the system's extensibility. As outlined in Fig. 1, user-defined types support relies on external Web services thus employing service-oriented architecture to validate and upload data from unknown documents and visualise them (see "API" and "UDT Support API" sections in the documentation). The architecture of the core system is based on the Model-View-Controller (MVC) pattern enabling good modularity and manageability.

Software engineering

Software engineering applied to Web application development together with mature open-source Microsoft ASP.Net Core software platform together with some additional components (Identity Manager, Dapper, Swagger, EPPlus, Serilog, FluentValidation) enabled flexibility and sustainability in development experience.

High-throughput experimentation

All information on the synthesis and characterisation of our thin film materials libraries can be found in ref. 10.

Data availability

No special data is required to reproduce the results of this research—only the source code, which is publicly available at https://gitlab.ruhr-uni-bochum.de/vic/infproject.

Code availability

Additional information is available from https://matinf.pro, public playground is available at https://inf.mdi.ruhr-uni-bochum.de.

Received: 18 October 2024; Accepted: 22 April 2025;
Published online: 03 May 2025

References

1. Brunton, S. L. & Kutz, J. N. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control* 1st edn (Cambridge University Press, Cambridge, 2019).
2. Scheffler, M. et al. FAIR data enabling new horizons for materials research. *Nature* **604**, 635–642 (2022). 2022.
3. Scheidgen, M. et al. NOMAD: a distributed web-based platform for managing materials science research data. *J. Open Source 1 Softw.* **8**, 5388 (2023).
4. Brandt, N. et al. Kadi4Mat: a research data infrastructure for materials science. *Data Sci. J.* **20**, 1–14 (2021).
5. Tremouilhac, P. et al. Chemotion repository, a curated repository for reaction information and analytical data. *Chemistry-Methods* **1**, 8 (2021).
6. Zakutayev, A. et al. An open experimental database for exploring inorganic materials. *Sci. Data* **5**, 1–12 (2018).
7. Mangwani, P., Mangwani, N. & Motwani, S. Evaluation of a multitenant saas using monolithic and microservice architectures. *SN Comput. Sci.* **4**, 185 (2023). 2023.
8. Giretti, A. Secure your application with OpenID Connect. In *Beginning gRPC with ASP.NET Core 6, edited by Joan Murray*, 1st edn. (Apress, Berkeley, CA, 2022).
9. Banko, L. & Ludwig, A. Fast-track to research data management in experimental material science – setting the ground for research group level materials digitalization. *ACS Comb. Sci.* **22**, 401–409 (2020). 2020.
10. Ludwig, A. Discovery of new materials using combinatorial synthesis and high-throughput characterization of thin-film materials libraries combined with computational methods. *npj Comput Mater.* **5**, 70 (2019). 2019.

Acknowledgements

This research was financially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Project-ID 388390466-TRR 247 (subproject INF). L.B. and A.L. also acknowledge funding from Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)–CRC 1625, project number 506711657, subproject INF.

Author contributions

V.D. has been a major contributor to *MatInf* development and was instrumental in writing the manuscript. L.B. provided materials science expertise and workflow formalisations for RDMS, and A.L. supervised and coordinated the research.

Funding

Open Access funding enabled and organized by Projekt DEAL.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41524-025-01618-1.

Correspondence and requests for materials should be addressed to Victor Dudarev, Lars Banko or Alfred Ludwig.