Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach

Paul-Olivier Dehaye[1] Mihnea Iancu[2] Michael Kohlhase[2] Alexander Konovalov[3] Samuel Lelièvre[4] Dennis Müller[2] Markus Pfeiffer[3] Florian Rabe[2] Nicolas M. Thiéry[4] Tom Wiesing[2]

[1] University of Zürich
[2] Jacobs University
[3] University of St Andrews
[4] Université Paris-Sud

Abstract. OpenDreamKit – "Open Digital Research Environment Toolkit for the Advancement of Mathematics" – is an H2020 EU Research Infrastructure project that aims at supporting, over the period 2015–2019, the ecosystem of open-source mathematical software systems. OpenDreamKit will deliver a flexible toolkit enabling research groups to set up Virtual Research Environments, customised to meet the varied needs of research projects in pure mathematics and applications.
An important step in the OpenDreamKit endeavor is to foster the interoperability between a variety of systems, ranging from computer algebra systems over mathematical databases to front-ends. This is the mission of the integration work package. We report on experiments and future plans with the *Math-in-the-Middle* approach. This architecture consists of a central mathematical ontology that documents the domain and fixes a joint vocabulary, or even a language, going beyond existing systems such as OpenMath, combined with specifications of the functionalities of the various systems. Interaction between systems can then be enriched by pivoting around this architecture.

1 Introduction

From their earliest days, computers have been used in pure mathematics to make tables, prove theorems (famously the four colour theorem) or, as with the astronomer's telescope, to explore new theories. Computer-aided experiments, and the use of databases relying on computer calculations such as the Small Groups Library in GAP, the Modular Atlas in group and representation theory, or the L-functions and Modular Forms Database (LMFDB, see later), are part of the standard toolbox of the pure mathematician. Certain areas of mathematics completely depend on these libraries. Computers are also increasingly used to support collaborative work and education.

In the last decades we witnessed the emergence of a wide ecosystem of open-source tools to support research in pure mathematics. This ranges from specialized to general purpose computational tools such as GAP, PARI/GP, LinBox, MPIR, Sage, or Singular, via online databases like the LMFDB or online services like Wikipedia, arXiv, to webpages like MathOverflow. A great opportunity is the rapid emergence of key technologies, in particular the Jupyter (previously IPython) platform for interactive and exploratory computing which targets all areas of science.

This has proven the viability and power of collaborative open-source development models, by users and for users, even for delivering general purpose systems targeting large audiences such as researchers, teachers, engineers, amateurs, and others. Yet some critical long term investments, in particular on the technical side, are in order to boost the productivity and lower the entry barrier:

– Streamlining access, distribution, portability on a wide range of platforms, including High Performance Computers or cloud services.
– Improving user interfaces, in particular in the promising area of collaborative workspaces as those provided by SageMathCloud.
– Lowering barriers between research communities and promoting dissemination. For example make it easy for a specialist of scientific computing to use tools from pure mathematics, and vice versa.
– Bringing together the developer communities to promote tighter collaboration and symbiosis, accelerate joint development, and share best practices.
– Structure the development to outsource as much of it as possible to larger communities, and focus manpower on core specialities: the implementation of mathematical algorithms and databases.
– And last but not least: Promoting collaborations at all scales to further improve the productivity of researchers in pure mathematics and applications.

OpenDreamKit – "Open Digital Research Environment Toolkit for the Advancement of Mathematics" [ODK] – is a project funded under the European H2020 Infrastructure call [EI] on *Virtual Research Environments*, to work on many of these problems.

In Section 2, we will introduce the OpenDreamKit project to establish the context for the "Math-in-the-Middle" (MitM) integration approach described in Section 3. The remaining sections then elucidate the approach by presenting first experiments and refinements of the chosen integration paradigm: Section 4 details how existing knowledge representation and data structures can be represented as MitM interface theories with a case study of equipping the LMFDB with a MitM-based programming interface. Section 5 discusses system integration between GAP and Sage and how this can be routed through a MitM ontology. Section 6 concludes the paper and discusses future work.

2 The OpenDreamKit project (2015-2019)

The OpenDreamKit project runs for four years, starting in September 2015, and involves about 50 people spread over 15 sites in Europe, with a total budget of 7.6 million euros. The largest portion of that is devoted to employing an average of 11 researchers and developers working full time on the project, while the other participants contribute the equivalent of six people working full time.

OpenDreamKit's goal is to develop *Virtual Research Environments* (VRE), that is online services enabling groups of researchers, typically spread across many countries, to work collaboratively on a per project basis. Rather than constructing a large monolithic VRE, we have designed our proposal around the long-term investments listed in the previous section, working on the large scale yet modular integration of mathematical software. Our goal is a modular, interoperable, and customisable VRE toolkit built out of relatively modest components, interfaced through our approach to work on the grease to make this work. According to the funding scheme, the project addresses, besides its technical goals, aspects such as outreach, dissemination, or tools to support teaching.

An innovative aspect of the OpenDreamKit project is that its preparation and management happens, as much as is practical and without infringing on privacy, in the open. For example, most documents, including the proposal itself, are version controlled on public repositories and progress on tasks and deliverables is tracked using public issues (see [ODK]). This has proven a strong feature to collaborate tightly with the community and get early feedback.

In practice, OpenDreamKit's work plan consists of several work packages: component architecture (modularity, packaging, distribution, deployment), user interfaces (Jupyter interactive notebook interfaces, 3D visualization, documentation tools), high performance mathematical computing (especially on multi-core/parallel architectures), a study of social aspects of collaborative software development, and a package on data/knowledge/software-bases.

The latter package focuses on the identification and extension of ontologies and standards to facilitate safe and efficient storage, reuse, interoperation and sharing of rich mathematical data, whilst taking provenance and citability into account. Its outcome will be a component architecture for semantically sound data archival and sharing, and integrate computational software and databases. The aim is to enable researchers to seamlessly manipulate mathematical objects across computational engines (e.g. switch algorithm implementations from one computer algebra system to another), front end interaction modes (database queries, notebooks, web, etc) and even backends (e.g. distributed vs. local).

In this paper, we discuss the general approach chosen to develop this semantically aware component architecture.

3 Integrating mathematical software systems via the Math-in-the-Middle approach

As discussed before, we aim to make our components interoperable at a mathematical level. In particular, we have to establish a common meaning space that will allow us to share computation, visualization of the mathematical concepts, objects, and models between the respective systems. This mediation problem is well understood in information systems [Wie92], and has for instance been applied to natural language translation via a hub language [KW03]. Here, our hub is mathematics itself, and the vocabulary (or even language) admits further formalisation that translates into direct gains in interoperability. For this reason, neither OpenMath [Bus+04] nor MathML [Aus+03] have the practical expressivity needed for our intended applications.

3.1 A common meaning space for interoperability

One problem is that the software systems in OpenDreamKit cover different mathematical concepts, and if there are overlaps, their models for them differ, and the implementing objects have different functionalities. This starts with simple naming issues (*e.g.* elliptic curves are named ec in the LMFDB, and as EllipticCurve in Sage), persists through the underlying data structures and in differing representations in the various tables of the LMFDB), and becomes virulent at the level of algorithms, their parameters, and domains of applicability.

To obtain a common meaning space for a VRE, we have the three well-known approaches in Figure 1.

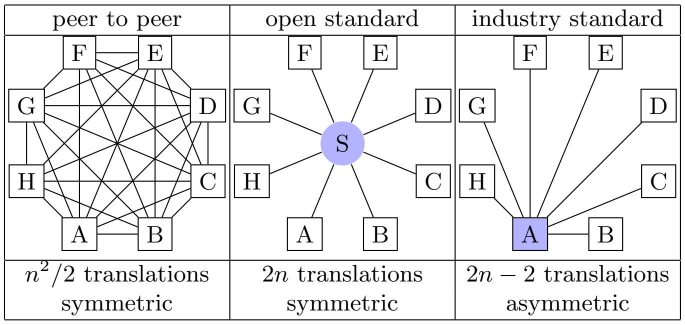

Fig. 1. Approaches for many-systems interoperability

The first does not scale to a project with about a dozen systems, for the third there is no obvious contender in the OpenDreamKit ecosystem. Fortunately, we already have a "standard" for expressing the meaning of mathematical concepts – **mathematical vernacular**: the language of mathematical communication, and in fact all the concepts supported in the OpenDreamKit VRE are documented in mathematical vernacular in journal articles, manuals, etc. The obvious problem is that mathematical vernacular is too *i*) *ambiguous*: we need a human to understand structure, words, and symbols *ii*) *redundant*: every paper introduces slightly different notions.

Therefore we explore an approach where we **flexiformalize**, i.e. partially formalize; see [Koh13] mathematical vernacular to obtain a flexiformal ontology of mathematics that can serve as an open communication vocabulary. We call the approach the **Math-in-the-Middle** (MitM) Strategy for integration and the ontology the **MitM ontology**.

Before we go into any detail on this ontology, and how it induces a uniform meaning space – see Section 4 for an example – we have to address another problem: the descriptions in the MitM ontology must simultaneously be system-near to make interfacing easy for systems, and serve as an interoperability standard – *i.e.* be general and stable. If we have an ontology system that allows modular/structured ontologies, we can solve this apparent dilemma by introducing **interface theories** [KRSC11], *i.e.* ontology modules (the light purple circles in Figure 2) that are at the same time system-specific in their description of mathematical concepts – near the actual representation of the system and part of the greater MitM ontology (depicted by the cloud in Figure 2) as they are connected to the core MitM ontology (the blue circle) by views we call **interface views**. The MitM approach stipulates that interface theories and interface views are maintained and released together with the respective systems, whereas the core MitM ontology represents the mathematical scope of the VRE and is maintained with it. In fact in many ways, the core MitM ontology is the conceptual essence of the mathematical VRE.

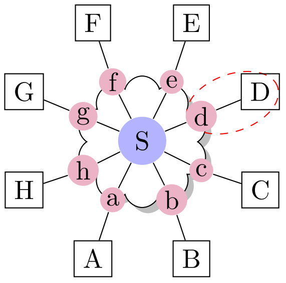

Fig. 2. Interface theories

3.2 Realizing and utilizing a MitM ontology

Our current candidate for representing the MitM ontology is the OMDoc/MMT format [Koh06; MMT]. OMDoc/MMT is an ontology format specialized for representing mathematical knowledge modularly in a theory graph: **theories** are collections of declarations of concepts, objects, and their properties that are connected by truth-preserving mappings called **theory morphisms**. The latter come in two forms: **inclusions** and **structures** that essentially correspond to object-oriented inheritance (direct inheritance and and inheritance modulo renaming and identification of symbols), and **view** that connect pre-existing theories – in these all axioms of the source theory have be to proven in the target theory. See [RK13] for a full account. Figure 3 shows an example of a theory graph. It has three layers:

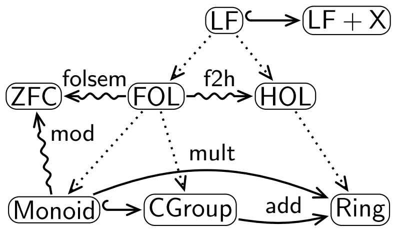

Fig. 3. A OMDoc/MMT Theory Graph

i) the (bottom) **domain level** which specifies mathematical domains as theories; here parts of elementary algebra. The hooked arrows are inclusions for inheritance, while the regular arrows are named structures that induce the additive and multiplicative structures of a ring.

ii) the **logic level** represents the languages we use for talking about the properties of the objects at the domain level – again as theories: the meta-theories of the domain-level ones – the dotted arrows signify the meta-relation. At this level, we also have inclusions and views (the squiggly arrows) which correspond to logic translations (f2h) and interpretations into **foundational theories** like set theory (here ZFC). Incidentally models can be represented as views into foundations.

iii) The top layer contains theories that act as metalogics, *e.g.* the Logical Framework LF and extensions which can be used to specify logics and their translations.

The theory graph structure is very well-suited to represent heterogeneous collections of mathematical knowledge, because views at the domain level can be used to connect differing but equivalent conceptualizations and views at the logic level can be used to bridge the different foundations of the various systems. The top level is only indirectly used in the MitM framework: it induces the joint meaning space via the meta-relation.

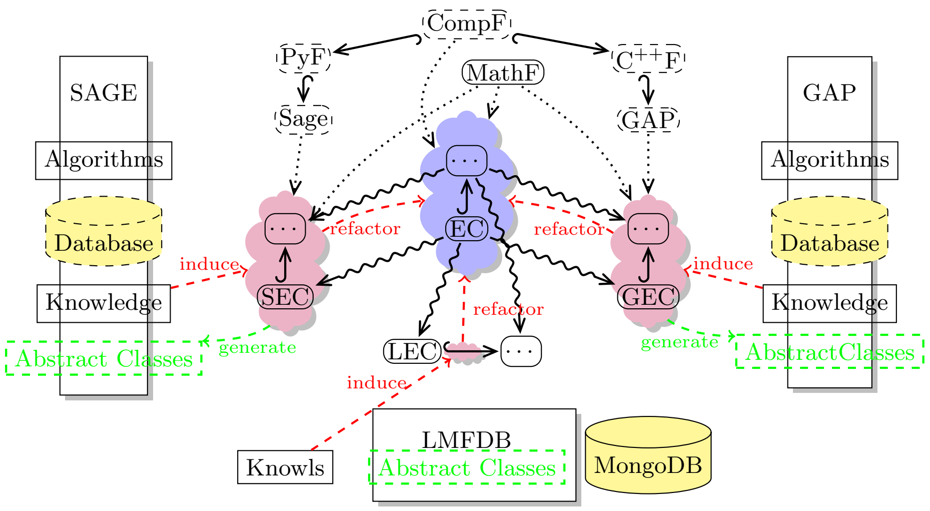

Fig. 4. The MitM paradigm in detail. PyF, C^{++}F and CompF are (basic) foundational theories for PYTHON, C^{++} and a generic computational model. SEC, LEC and GEC are theories for SAGE, LMFDB and GAP elliptic curves.

If we apply OMDoc/MMT to the MitM architecture, we arrive at the situation in Figure 4, where we drill into the MitM information architecture from Figure 2, but restrict at this stage to three systems from the OPENDREAMKIT project. In the middle we see the core MitM ontology (the blue cloud) as an OMDoc/MMT theory graph connected to the interface theories (the purple clouds) via MitM interface views. Conceptually, the systems in OPENDREAMKIT consist of three main components:

i) a *Knowledge Representation component* that provides data structures for the objects modeling mathematical concepts and their properties.

ii) a *DataBase component* that provides mass storage for objects, and

iii) a *library of algorithms* that operate on these.
To connect a system to an MitM-based VRE, the knowledge representation component is either refactored so that it can generate interface theories, or a schema-like description of the underlying data structures is created manually from which abstract data structures for the system can be generated automatically – in this version the interface theories act as an Interface Description Language.

In this situation there are two ways to arrive at a greater MitM ontology: the OpenDreamKit project aims to explore both: either *i*) standardizing a core MitM by refactoring the various interface theories where they overlap, or *ii*) flexiformalizing the available literature for a core MitM ontology. For *i*), the MitM interface views emerge as refinements that add system-specific details to the general mathematical concepts[1] For *ii*), we have to give the interface views directly.

To see that this architecture indeed gives us a uniform meaning space, we observe that the core MitM ontology uses a mathematical foundation (presumably some form of set theory), whereas the interface theories also use system-specific foundations that describe aspects of the computational primitives of the respective systems. We have good formalizations of the mathematical foundations already; first steps towards a computational ones have been taken in [KMR13].

Our efforts also fit neatly alongside similar efforts underway across the sciences to standardize metadata formats (for instance through the Research Data Alliance's Typing Registry Working Group [Rda]), except for a typically much higher complexity in the typing since our objects of study are sometimes seen as types and sometimes as instances (think of groups for instance).

4 LMFDB knowledge and interoperability

The *L-functions and modular forms database* is a project involving dozens of mathematicians who assemble computational data about L-functions, modular forms, and related number theoretic objects. The main output of the project is a website, hosted at `http://www.lmfdb.org`, that presents this data so that it can serve as a reference for research efforts, and is accessible for postgraduate students. The mathematical concepts underlying the LMFDB are extremely complex and varied, so part of the effort has been focused on how to relay knowledge, such as mathematical definitions and their relationships, to data and software. For this purpose, the LMFDB has developed so-called *knowls*, which are a technical solution to present LaTeX-encoded information interactively, heavily exploiting the concept of transclusion. The end result is a very modular and highly interlinked set of definitions in mathematical vernacular which can be

[1] We use the word "interface theory" with a slightly different intention when compared to the original use in [KRSC11]: There the core MitM ontology would be an interface between the more specific implementations in the systems, whereas here we use the "interface theories" as interfaces between systems and the core MitM ontology. Technically the same issues apply.

easily anchored in vastly different contexts, such as an interface to a database, to browsable data, or as constituents of an encyclopedia [Lmfc].

The LMFDB code is primarily written in PYTHON, with some reliance on SAGE for the business logic. The backend uses the NoSQL document database system MONGODB [Lmfa]. Again, due to the complexity of the objects considered, many idiosyncratic encodings are used for the data. This makes the whole data management lifecycle particularly tricky, and dependent on different select groups of individuals for each component.

As the LMFDB spans the whole "vertical" workflow, from writing software, to producing new data, up to presenting this new knowledge, it is a perfect test case for a large scale case study of the MitM approach. Conversely, a semantic layer would be beneficial to its activities across data, knowledge and software, which it would help integrate more cohesively and systematically.

Among the components of the LMFDB, elliptic curves stand out in the best shape, and a source of best practices for other areas. We have generated MitM interface theories for LMFDB elliptic curves by (manually) refactoring and flexiformalizing the LaTeX source of knowls into sTeX (see Listing 1.1 for an excerpt), which can be converted into flexiformal OMDoc/MMT automatically. The MMT system can already type-check the definitions, avoiding circularity and ensuring some level of consistency in their scope and make it browsable through MathHub.info, a project developed in parallel to MMT to host such formalisations.

Listing 1.1. sTeX flexiformalization of an LMFDB knowl (original: [Lmfd])

```
\begin{mhmodnl}{minimal−Weierstrass−model}{en}
    A \defi{minimal} \trefii{Weierstrass}{model} is one for which
    $\absolutevalue\wediscriminantOp$ is minimal among all Weierstrass models
    for the same curve. For elliptic curves over $\RationalNumbers$, minimal
    models exist, and there is a unique minimal model which satisfies the
    additional constraints $\minset{\livar{a}1,\livar{a}3}{\set{0,1}}$, and
    $\inset{\livar{a}2}{\set{−1,0,1}}$.
    This is defined as the reduced minimal Weierstrass model of the elliptic curve.
  \end{definition}
\end{mhmodnl}
```

The second step consisted of translating these informal definitions into progressively more exhaustive MMT formalisations of mathematical concepts (see Listing 1.2). The two representations are coordinated via the theory and symbol names – we can see the sTeX representation as a human-oriented documentation of the MMT.

Listing 1.2. MMT formalisation of elliptic curves and their Weierstrass models

```
theory minimal_Weierstrass_model : odk:?Math =
  include ?elliptic_curve D
  minimal : tm Ws_model → tm Ws_model D
  is_minimal : tm Ws_model → prop \US = [A] (minimal A) ≐ A D
  minimality_idempotence : {A} ⊢ minimal (minimal A) ≐ minimal A D
  minimality_of_minimal_Ws_model :
    {A} ⊢ is_minimal (minimal_Ws_model A) D
  injective_minimal_Ws_model :
    {A,B} ⊢ minimal_Ws_model A ≐ minimal_Ws_model B → ⊢ A ≐ B D
M
```

Finally, we have to integrate computational data into the interface theories. Based on recent ongoing efforts [Lmfb] to document the LMFDB “data schemata” we established OMDoc/MMT theories that link the database fields to their data types (string *vs.* float *vs.* integer tuple, for instance) and mathematical types (elliptic curves or polynomials) – the latter based on the vocabulary in the interface theories generated from the LMFDB knowls. This schema theory is complemented by a theory on functorial hence composable *MMT codecs*, which in turn acts as a specification for a collection of implementations in various programming languages (currently PYTHON, Scala, and C^{++} for SAGE, MMT, and GAP respectively) which are first instances of a computational foundation (see Section 3). For instance, one can compose two MMT codecs, say *polynomial-as-reversed-list* and *rational-as-tuple-of-int*, to signify that the data $[(2,3),(0,1),(4,1)]$ is meant to represent the polynomial $4x^2 + 2/3$. Of course, these codecs could be further decomposed (e.g. for signaling which variable name to use). The initial cost of developing these codecs is high, but the clarity gained in documentation is valuable, they are highly reusable, and they drastically expand the range of tooling that can be built around data management.

A typical application Based on these MitM interface theories we can generate I/O interfaces that translate between the low-level LMFDB API, which delivers raw MONGODB data in JSON format into MMT expressions that are grounded in the interface theories. This ties the LMFDB database into the MitM architecture transparently. As a side effect, this opens up the LMFDB to programmatic queries via the MMT API, which can be queried and can then relay them to the LMFDB API directly and transparently.

5 Distributed collaboration with GAP/Sage

Another aspect of interoperability in a mathematical VRE is the possibility of distributed multisystem computations, where *e.g.* a given system may decide to delegate certain subcomputations or reasoning tasks to other systems.

There are already a variety of peer-to-peer interfaces between systems in the OPENDREAMKIT project (see Figure 1), which are based on the *handle paradigm*; for example SAGE includes, among others, interfaces for GAP, SINGULAR, and PARI. In this paradigm, when a system A delegates a calculation to a system B, the result r of the calculation is not converted to a native A object; instead B just returns a *handle* h (or reference) to the object r. Later, A can run further calculations with r by passing it as argument to functions or methods implemented by B. Some advantages of this approach are that we can avoid the overhead of back and forth conversions between A and B, and that we can manipulate objects of B from A, even if they have no native representation in A.

The next desirable feature is for the handle h to behave in A as if it was a native A object; in other words, one wants to adapt the API satisfied by r in B to match the API for the same kind of objects in A. For example, the method call `h.cardinality()` on a SAGE handle `h` to a GAP object `G` should trigger in GAP the corresponding function call `Size(G)`.

This can be implemented using the classical *adapter pattern*, mapping calls to SAGE's method to corresponding GAP methods. *Adapter classes* have already been implemented for certain types of objects, like SAGE's `PermutationGroup` or `MatrixGroup`. However, this implementation lacks modularity: for example, if `h` is a handle to a mere set `S`, SAGE cannot use the *adapter method* that maps `h.cardinality()` to `Size(S)`, because this adapter method is only available in the above two adapter classes.

To get around this problem we have worked on a more semantic integration, where adapter methods are made aware of the type hierarchies of the respective systems, and defined at the highest available level of generality, as in Listing 1.3.

Listing 1.3. A semantic adapter method in SAGE

```
class Sets: # Everything generic about sets in Sage
    class GAP: # Adapter methods relevant to Sets in the Sage−Gap interface
        class ParentMethods: # Adapter methods for sets
            def cardinality(self): # The adapter for the cardinality method
                return self.gap().Size().sage()
        class ElementMethods: # Adapter methods for set elements
            ...
        class MorphismMethods: # Adapter methods for set morphisms
            ...
class Groups: # Everything generic about groups in Sage
    # This automatically includes features defined at a more general level
```

This peer-to-peer approach however does not scale up to a dozen systems. This is where the MitM paradigm comes to the rescue. With it, the task is reduced to building interface theories and interface views into the core MitM ontology in such a way that the adapter pattern can be made generic in terms of the MitM ontology structure, without relying on the concrete structure of the respective type systems. Then the adapter methods for each peer-to-peer interface can be automatically generated. In our example the adapter method for `cardinality` can be constructed automatically as soon as the MitM interface views link the `cardinality` function in the SAGE interface theory on Sets with the `Size` function in the corresponding interface theory for GAP.

We will now show first results of our experiments with interface theories and interface views, including several applications beyond the generation of interface theories that support distributed computation for SAGE and GAP.

5.1 Semantics in the Sage category system

The Sage library includes 40k functions and allows for manipulating thousands of different kinds of objects. In any large system it is critical to tame code bloat by

i) identifying the core concepts describing common behavior among the objects;

ii) implementing generic operations that apply on all objects having a given behavior, with appropriate specializations when performance calls for it;

iii) designing or choosing a process for selecting the best implementation available when calling an operation objects.

Following mathematical tradition and the precedent of the Axiom, Fricas, or MuPAD systems, Sage has developed a category-theory-inspired “category system”, and found a way to implement it on top of the underlying Python object system [Dev16; SC]. In short, a **category** specifies the available **operations** and the **axioms** they satisfy. This category system models taxonomic knowledge from mathematics explicitly and uses it to support genericity, control the method selection process, structure the code and documentation, enforce consistency, and provide generic tests.

To generate interface theories from the Sage category system, we are experimenting with a system of annotations in the Sage source files. Consider for instance the situtation in Figure 5 where we have annotated the `Sets()` category in Sage with `@semantic` lines that state correspondences to other interface theories. From these the Sage-to-MMT exporter can generate the respective interface theories and views.

```
@semantic(mmt="sets")
class Sets:
    class ParentMethods:
        @semantic(mmt="card?card", gap="Size")
        @abstractmethod
        def cardinality(self):
            "Return the cardinality of ``self``"
```

Fig. 5. An annotated category in Sage

In ongoing experiments, variants of the annotations are tested for annotating existing categories without touching their source files and providing the signature or the corresponding method names in other systems when this information has not yet been formalized elsewhere.

5.2 Exporting the GAP knowledge: type system documentation

As in Sage, the GAP type system encodes a wealth of mathematical knowledge, which can influence method selection. For example establishing that a group is nilpotent will allow for more efficient methods to be run for finding its centre. The main difference between Sage and GAP lies in the method selection process. In Sage the operations implemented for an object and the axioms they satisfy are specified by its class which, together with its super classes, groups syntactically all the methods applicable in this context. In GAP, this information is instead specified by the truth-values of a collection of independent **filters**, while the context of applicability is specified independently for each method. Breuer and Linton describe the GAP type system in [BL] and the GAP documentation [Gap] also contains extensive information on the types themselves.

GAP allows some introspection of this knowledge after the system is loaded: the values of those attributes and properties that are unknown on creation, can be computed on demand, and stored for later reuse.

As a first step in generating interface theories for the MitM ontology, we have developed tools to access mathematical knowledge encoded in GAP, such as introspection inside a running GAP session, export to JSON to import to MMT, and export as a graph for visualisation and exploration. These will become generally available in the next GAP release. The JSON output of the GAP object system with default packages is currently around 11 Megabytes, and represents a knowledge graph with 540 vertices, 759 edges and 8 connected components, (see Figures 6,7). If all packages are loaded, this graph expands to 1616 vertices, 2178 edges and 17 connected components.

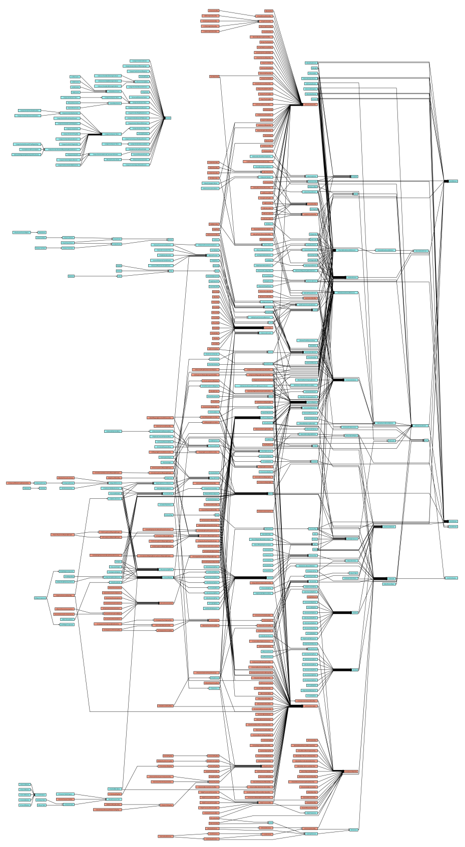
Fig. 6. The GAP Knowledge Graph.

There is, however, another source of knowledge in the GAP universe: the documentation, which is provided in the GAPDoc format [LN12]. Besides the main manuals, GAPDoc is adopted by 97 out of the 130 packages currently redistributed with GAP. Conventionally GAPDoc is used to build text, PDF and HTML versions of the manual from a common source given in XML. The reference manual has almost 1400 pages and the packages add hundreds more.

The GAPDoc sources classify documentation by the type of the documented object (function, operation, attribute, property, etc.) and index them by system name. In this sense they are synchronized with the type system (which *e.g.* has the types of the functions) and can be combined into flexiformal OMDoc/MMT interface theories, just like the ones for LMFDB in Section 4. This conversion is currently under development and will lead to a significant increase of the scope of the MitM ontology.

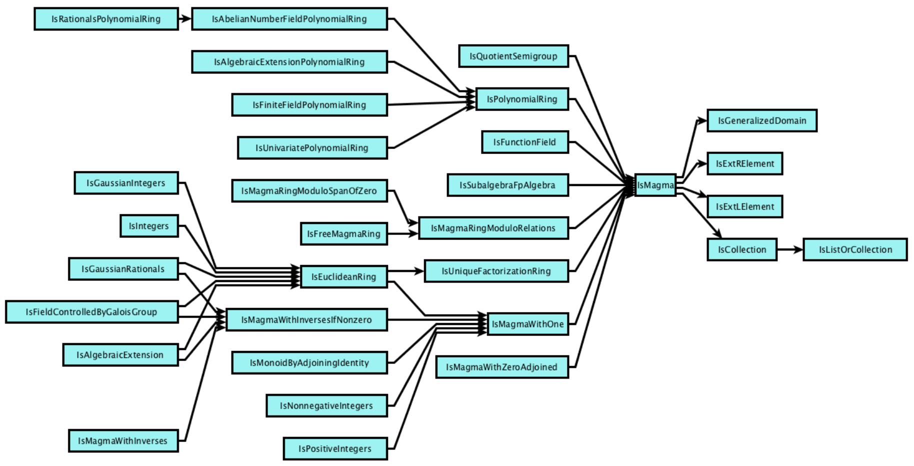

Fig. 7. The GAP Knowledge Graph (fragment).

As a side-effect of this work, we discovered quite a few inconsistencies in the GAP documentation which came from a semi-automated conversion of GAP manuals from the TEX-based manuals used in GAP 4.4.12 and earlier. We developed the consistency checker for the GAP documentation, which extracts type annotations from the documented GAP objects and compares them with their actual types. It immediately reported almost 400 inconsistencies out of 3674 manual entries, 75% of which have been eliminated in the subsequent cleanup.

6 Conclusion

In this paper we have presented the OpenDreamKit project and the "Math-in-the-Middle" approach it explores for mitigating the system integration problems inherent in combining an ecosystem of open source software systems into a coherent mathematical virtual research environment. The MitM approach relies on a central, curated, flexiformal ontology of the mathematical domains to be covered by the VRE together with system-near interface theories and interface views to the core ontology that liaise with the respective systems. We have reported on two case studies that were used to evaluate the approach: an interface for the LMFDB, and a more semantic handle interface between GAP and Sage.

Even though the development of the MitM is still at a formative stage, these case studies show the potential of the approach. We hope that the nontrivial cost of curating an ontology of mathematical knowledge and interface views to the interface theories will be offset by its utility as a resource, which we are currently exploring; the unification of the knowledge representation components

1. enables VRE-wide domain-centered (rather than system-centered) documentation;

2. can be leveraged for distributed computation via uniform protocols like the SCSCP [HR09] and MONET-style service matching [CDT04] (the absence of content dictionaries – MitM theories – was the main hurdle that kept these from gaining more traction);
3. will lead to the wider adoption of best practices in mathematical knowledge management in the systems involved; in fact, this is already happening.

Whether in the end the investment into the MitM will pay off also depends on the quality and usability of the tools for mathematical knowledge management. Therefore we invite the CICM community to interact with and contribute to the OpenDreamKit project, on this work package and the others. Possible contributions include

1. interfacing another system to the MitM architecture via interface theories
2. contributing to the MitM core ontology
3. MitM-refactoring existing integrations of mathematical software systems.

Acknowledgements The authors gratefully acknowledge the other participants of the St Andrews workshop, in particular John Cremona, Luca de Feo, Steve Linton, and Viviane Pons, for discussions and experimentation which clarified the ideas behind the math-in-the-middle approach.

We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541), from the EPSRC Collaborative Computational Project CoDiMa (EP/M022641/1) and from the Swiss National Science Foundation grant PP00P2_138906.

References

[Aus+03] R. Ausbrooks et al. "Mathematical Markup Language (MathML) v. 2.0." In: *World Wide Web Consortium recommendation* (2003).

[BL] Thomas Breuer and Steve Linton. "The GAP 4 Type System: Organising Algebraic Algorithms". In: *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation.* ISSAC '98. ACM, pp. 38–45.

[Bus+04] Stephen Buswell et al. *The Open Math standard.* Tech. rep. Version 2.0. The Open Math Society, 2004.

[CDT04] Olga Caprotti, Mike Dewar, and Daniele Turi. *Mathematical Service Matching Using Description Logic and OWL.* Tech. rep. The MONET Consortium, 2004.

[Dev16] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.0).* 2016. URL: `http://www.sagemath.org`.

[EI] *EINFRA-9: e-Infrastructure for Virtual Research Environment.* URL: `http://ec.europa.eu/research/participants/portal/desktop/en/opportunities/h2020/topics/2144-einfra-9-2015.html`.

[Gap] *GAP – Groups, Algorithms, and Programming, Version 4.8.2.* The GAP Group. 2016. URL: `http://www.gap-system.org`.

[HR09] P. Horn and D. Roozemond. “OpenMath in SCIEnce: SCSCP and POPCORN”. In: *MKM/Calculemus Proceedings*. Ed. by Jacques Carette et al. LNAI 5625. Springer Verlag, July 2009, pp. 474–479.

[KMR13] Michael Kohlhase, Felix Mance, and Florian Rabe. “A Universal Machine for Biform Theory Graphs”. In: *Intelligent Computer Mathematics*. Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013. DOI: `10.1007/978-3-642-39320-4`.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: `http://omdoc.org/pubs/omdoc1.2.pdf`.

[Koh13] Michael Kohlhase. “The Flexiformalist Manifesto”. In: *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*. Ed. by Andrei Voronkov et al. Timisoara, Romania: IEEE Press, 2013, pp. 30–36.

[KRSC11] Michael Kohlhase, Florian Rabe, and Claudio Sacerdoti Coen. “A Foundational View on Integration Problems”. In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 107–122.

[KW03] H. Kanayama and H. Watanabe. “Multilingual translation via annotated hub language”. In: *MT-Summit IX*. 2003, pp. 202–207.

[Lmfa] *LMFDB GitHub repository*. URL: `https://github.com/LMFDB/lmfdb`.

[Lmfb] *LMFDB inventory GitHub repository*. URL: `https://github.com/LMFDB/lmfdb-inventory`.

[Lmfc] *LMFDB Knowledge Database*. URL: `http://lmfdb.org/knowledge/`.

[Lmfd] *LMFDB Knowledge Database entry for* Minimal Weierstrass equation over the rationals. URL: `http://lmfdb.org/knowledge/show/ec.q.minimal_weierstrass_equation`.

[LN12] F. Lübeck and M. Neunhöffer. *GAPDoc, A Meta Package for GAP Documentation, Version 1.5.1*. 2012. URL: `http://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc`.

[MMT] Florian Rabe. *The MMT Language and System*. URL: `https://svn.kwarc.info/repos/MMT/doc/html` (visited on 10/11/2011).

[ODK] *OpenDreamKit Open Digital Research Environment Toolkit for the Advancement of Mathematics*. URL: `http://opendreamkit.org`.

[Rda] *Research Data Alliance Type Registries Working Group*. URL: `https://rd-alliance.org/groups/data-type-registries-wg.html`.

[RK13] Florian Rabe and Michael Kohlhase. “A Scalable Module System”. In: *Information & Computation* 0.230 (2013), pp. 1–54.

[SC] Nicolas M. Thiéry et al. *Elements, parents, and categories in Sage: a primer*. URL: `http://combinat.sagemath.org/doc/reference/categories/sage/categories/primer.html`.

[Wie92] Gio Wiederhold. “Mediators in the architecture of future information systems”. In: *Computer* 25.3 (1992), pp. 38–49.