REGULAR-T1: Knowledge-Based Interoperability for Mathematical Software Systems

Michael Kohlhase[1] Dennis Müller[1] Markus Pfeiffer[3] Florian Rabe[2]
Nicolas M. Thiéry[4] Victor Vasilyev[3] Tom Wiesing[1]

[1] FAU Erlangen-Nürnberg
[2] Jacobs University Bremen
[3] University of St Andrews
[4] Université Paris-Sud

Abstract. There is a large ecosystem of mathematical software systems. Individually, these are optimized for particular domains and functionalities, and together they cover many needs of practical and theoretical mathematics. However, each system specializes on one area, and it remains very difficult to solve problems that need to involve multiple systems. Some integrations exist, but the are ad-hoc and have scalability and maintainability issues. In particular, there is not yet an interoperability layer that combines the various systems into a virtual research environment (VRE) for mathematics.
The OpenDreamKit project aims at building a toolkit for such VREs. It suggests using a central system-agnostic formalization of mathematics (Math-in-the-Middle, MitM) as the needed interoperability layer. In this paper, we conduct the first major case study that instantiates the MitM paradigm for a concrete domain as well as a concrete set of systems. Specifically, we integrate GAP, SageMath, and Singular to perform computation in group and ring theory.
Our work involves massive practical efforts, including a novel formalization of computational group theory, improvements to the involved software systems, and a novel mediating system that sits at the center of a star-shaped integration layout between mathematical software systems.

1 Introduction

There is a large and vibrant ecosystem of open-source mathematical software systems. These range from calculators, which are only capable of performing simple computations, via mathematical databases, which curate collections of a mathematical objects, to powerful modeling tools and computer algebra systems (CAS).

Most of these systems are very specific – they focus on one or very few aspects of mathematics. For example, among databases, the "Online Encyclopedia of Integer Sequences" (OEIS) focuses on sequences over $\mathbb{Z}$ and their properties, and the "L-Functions and Modular Forms Database" (LMFDB) [Cre16;

LMFDB] on objects in number theory pertaining to Langland's program. Among CAS, GAP [GAP] excels at discrete algebra with a focus on group theory, Singular [SNG] focuses on polynomial computations with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory, and SageMath [Sage] aims to be a general purpose software for computational pure mathematics by loosely integrating many systems including the aforementioned ones.

For a mathematician, however, (a user, which we call Jane) the systems themselves are not relevant. Instead, she only cares about being able to solve problems. Because it is typically not possible to solve a mathematical problem using only a single program, Jane has to work with multiple systems and combine the results to reach a solution. Currently there is very little tool support for this practice, so Jane has to isolate sub-problems that the respective systems are amenable to, formulate them in the respective input language and collect intermediate results and reformulate them for the next system — a tedious and error-prone process at best, a significant impediment to scientific progress at worst. Solutions for some situations certainly exist, which can help get Jane unstuck, but these are ad-hoc and only for specific often-used system combinations. Moreover, each of these ad hoc solutions requires a lot of maintenance and scales badly to multi-system integration.

The OpenDreamKit project was initiated to tackle these problems systematically and build virtual research environments (VRE) on top of the existing systems. To build a VRE from individual systems, we need a joint user interface – the OpenDreamKit project adopts Jupyter [Jup] and active documents [Koh+11] – and an interoperability layer that allows passing problems and results between the disparate systems. For the latter, it proposes the Math-in-the-Middle (MitM [Deh+16]) paradigm, an interoperability framework based on a central, system-independent ontology of mathematical knowledge.

In this paper we instantiate the MitM paradigm in a concrete case study using a distributed computation involving GAP, SageMath, and Singular. We will use the following running example from computational group theory: Jane works in the polynomial ring $R = \mathbb{Z}[X_1, \ldots, X_4]$. She wants to compute the orbit $O = Orbit(D_8, R, X_1 + X_2) \subseteq R$ of the polynomial $X_1 + X_2 \in R$ under the action of the dihedral group[1] D_8, and then compute a Gröbner base $G = Groebner(Ideal(O)) \subseteq R$ for the ideal generated by O. Jane is a SageMath user and wants to receive the result in SageMath, but she wants to use GAP's orbit algorithm and Singular's Gröbner base algorithm, which she knows to be very efficient.

In Section 2 we recap the MitM paradigm. MitM solutions consists of three parts: a central ontology, specifications of the abstract languages of the involved systems, and the distributed computation infrastructure that connects the systems via the ontology as an intermediate representation. The rest of the paper

[1] Incidentally, this group is called D_4 in SageMath but D_8 in GAP due to differing conventions in different mathematical communities – a small example of the obstacles to system interoperability that MitM tackles.

develops these three parts for our case study: In Section 3, we contribute a fragment to the MitM ontology that formalizes computational group theory. In Section 4, we specify the abstract languages of GAP, SageMath, and Singular and their relation to the ontology. And in Section 5, we present the resulting virtual research environment built on these systems in action. Section 6 concludes the paper and compares MitM-based interoperability with other approaches.

2 Math-in-the-Middle Interoperability

Figure 1 shows the basic MitM design. We want to make the systems A to H interoperable. A P2P translation regime (n^2 translations between n systems) is already intractable for the systems in the OpenDreamKit project (more than a dozen). Alternatively, an "industry standard" regime, where one system's language is declared as the standard is infeasible, since no system subsumes the others in terms of coverage – not to mention the political problems such a standardization would induce. Instead, MitM uses a central mathematical ontology that provides an independent mediating vocabulary, via which all participating systems are aligned. All mathematical knowledge shared between the systems and exposed to the high-level VRE user is expressed using the vocabulary of this ontology.

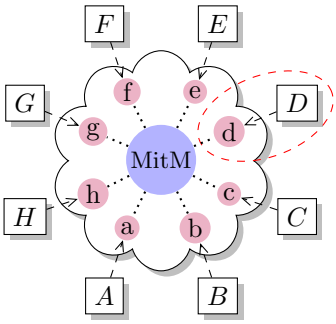

Fig. 1. MitM Paradigm

The following sections describe the three components of the MitM paradigm in more detail.

2.1 The MitM Ontology

In the center, we have the **MitM Ontology**, which is a formalization of the mathematical knowledge behind the systems A to H as a theory graph in the OMDoc/MMT format [Koh06; RK13; MMT]. We do not go into the details of OMDoc/MMT here. For our purposes, it suffices to assume that a theory graph formalizes a language for mathematical objects as a set of typed symbols with a (formal or informal) specification of their semantics. For example, the symbol `PolynomialRing` takes a ring of coefficients r and a number n of variables and returns the ring $r[X_1, \ldots, X_n]$ of polynomials.

Note that the purpose of the MitM ontology is not the formal verification of mathematical theorems (as for most formalizations such as [Gon+13] for group theory), but to act as a pivot point for integrating systems. This means that it can be much nearer to the informal but rigorous presentation of mathematical knowledge in the literature. While each system makes compromises and optimizations needed for a particular application domain, the MitM ontology follows existing and already informally standardized mathematical knowledge and can thus serve as a standard interface layer between systems

Importantly, the MitM ontology does not have to include any definitions or proofs — it only has to declare the types of all relevant symbols and state (but not prove) the relevant theorems. This makes it possible for users like Jane to write and extend MitM ontologies quickly, whereas other formalizations usually require extensive efforts by specialists.

2.2 Specifying Individual System Dialects

System Dialects It is unavoidable, that each system induces its own language for mathematical objects. This is the cause of much incompatibility because even subtle differences make naive integration impossible. Moreover, due to the difficulty of the involved mathematics and the effort of maintaining the implementations, such differences are aplenty.

Fortunately, we can at least easily abstract from the user-facing surface syntax of these languages: scalable interoperability can anyway only be achieved by acting on the internal data structures of the systems. Thus, only the much simpler internal abstract syntax needs to be considered.

The symbols that build the abstract syntax trees can be split into two kinds: *constructors* build primitive objects without involving computation, and *operations* compute objects from other objects (including predicates, which we see as operations that return booleans). For purposes of interoperability it is possible to abstract from this distinction and consider both as typed symbols. This abstraction is important because systems often disagree on the choice of primitive objects. Thus, we can represent the interfaces of the systems A to H as OMDoc/MMT theory graphs a to h that declare the constructors and operations (but omit all implementations of the operations) of the respective system.

Given the theory graph a, we can express all objects in the language of A as OMDoc/MMT objects using the symbols of a. We call this language the **OMDoc/MMT system dialect** for A and refer to these objects as A-objects. It is relatively straightforward to write (or even automatically generate) the theory graph a and to implement a serializer and parser for A-objects as a part of A. This is because no consideration of interoperability and thus no communication with the developers of other systems is needed.

Alignments with the Ontology The above reduces the interoperability problem to relating each system dialect to the MitM ontology. Each system dialect overlaps with the language of the ontology, but no system implements all ontology symbols and every systems implements many idiosyncratic operations that are not part of the ontology. Some system symbols are related to corresponding symbols in the MitM ontology. Then we can use the MitM ontology as an intermediate representation to bridge between any two systems, e.g., by translating A-objects to the corresponding ontology objects and then those to the corresponding B-objects.

However, even when A and B deal with the "same mathematical objects", these may be constructed and represented differently, e.g., symbols can differ in

name, argument order/number, types, etc. A major difficulty for system interoperability is bridging these differences. To formalize the details of this relation, [Mül+17b] introduced **OMDoc/MMT alignments**. These are essentially pairs of OMDoc/MMT symbol identifiers decorated by a set of key-value pairs. Whenever two systems' APIs declare symbols for the same mathematics operation, the alignments with the MitM ontology determine which ontology objects correspond to system objects.

The alignment of a-symbols to ontology symbols must be spelled out manually. But this is usually straightforward and easy even for inexperienced users.

Thus we can reduce the problem of interfacing n systems to *i*) curating a MitM ontology for the joint mathematical domain, *ii*) generating n theory graphs for system dialects, *iii*) maintaining n collections of alignments into the MitM ontology.

Alignments form an independent part of the MitM interoperability infrastructure. Incidentally they obey a distinct development schedule: the MitM ontology will be under continuous development — it formalizes the common knowledge of the community about a mathematical domain. The system dialects are released together with the systems according to their respective development cycle. The alignments bridge between them and have to mediate these cycles.

2.3 MitM-based Distributed Computation

The final missing piece for a system interoperability layer for a VRE toolkit is a practical way of transporting objects between systems. This requires two steps.

Firstly, if the system theory graphs and alignments are known, we can automatically translate A-objects to B-objects in two steps: A to ontology and ontology to B. This two-step translation has been implemented in [Mül+17a] based on the MMT system [Rab13; MMT] – an implementation of the OMDoc/MMT format and a set of mathematical knowledge management algorithms.

Secondly, each system A has to be able to serialize/parse A-objects and to send them to/receive them from MMT. In the OpenDreamKit project we use the OpenMath SCSCP (Symbolic Computation Software Composability) protocol [Fre+] for that. It is straightforward to extend a parser/serializer for A-objects to an SCSCP clients/server by implementing the SCSCP protocol on top of, e.g., sockets or using an existing SCSCP library.

3 The MitM Ontology for Computational Group Theory

Jane's use case involves groups and actions, polynomials, rings and ideals, and Gröbner bases, all of which must be formalized in the MitM ontology. Due to space restrictions, we only describe the ontology for computational group theory (CGT) as an example. This formalization can be found at [Mitb].

3.1 Type Theory and Logic

OMDoc/MMT formalizations must be relative to foundational logic, which is itself formalized in OMDoc/MMT. As foundation for all formalizations in MitM [Mita], we use a polymorphic dependently typed λ-calculus with two universes type and kind (roughly analogous to sets and proper classes in set theory) and subtyping. It provides dependent function types {a:A}B(a), representing the type of all functions mapping an argument a:A to some element of type B(a). If B does not depend on the argument a, we obtain the simple function type A→B.

For formulas, we use a type prop and a higher order logic where quantifiers range over any type. We furthermore follow the judgments-as-type paradigm by declaring a function ⊢:prop→type mapping propositions to the *type of their proofs*, which allows us to declare proof rules as functions mapping proofs (of the premises) to a proof (of the conclusion).

The judgment A<:B expresses that A is a subtype of B. We use power types (the type of subtypes of a type) and predicate subtyping {'a:A | P(a)'}. The latter makes type-checking undecidable, but that is necessary for natural formalizations in many areas of mathematics.

Additionally we extend our type theory with record types, which is critical for formalizing mathematical structures. In particular,ModelsOf T is the record type of models of the theory T. This lets us, e.g., define groups by the theory of operations and its signature and axioms, while group=ModelsOf group_theory is the type of all models of said theory, i.e., all groups, as seen in Figure 2. Any element g:group thus represents an actual group, whose operations and axioms can be accessed via record field projections (e.g. g.inverse yields the inverse operation of g. Since axioms are turned into record type fields as well, actually constructing a record of type group corresponds to proving that the field universe and the operations provided in the record do in fact form a group.

```
theory group : base:?Logic =
  theory group_theory : base:?Logic =
    include ?monoid/monoid_theory

    inverse : U → U  # 1 ⁻¹  prec 24
    inverseproperty : ⊢ ∀ [x] x ∘ x ⁻¹ ≐ e
  ❙
  group = ModelsOf group_theory
```

Fig. 2. MitM ontology Fragment

3.2 Group Theory

Our formalization of CGT follows the template of its implementation in GAP, and requires different levels of abstraction, currently *abstract*, *representation*, *implementation*, and *concrete*. From our experience, we expect this pattern to be applicable across computational algebra, possibly with additional levels of abstraction. The left box in Figure 3 shows the levels and their relation to the constructors and operations of GAP.

Abstract Level This contains the theory of *Groups*: the group axioms, generating sets, homomorphisms, group actions, stabilisers, and orbits. This also easily leads into definitions of centralisers – i.e. stabilisers of elements under conjugation

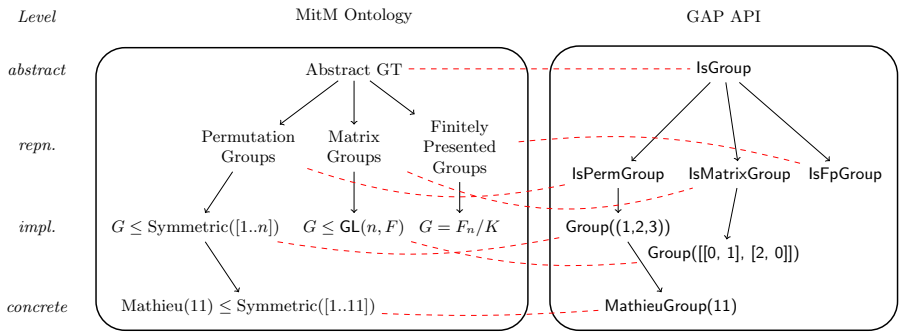

Fig. 3. Alignments between the MitM Ontology and the GAP API

– and normalisers – i.e. stabilisers of subgroups under conjugation, stabiliser chains, Sylow-p subgroups, Hall subroups, and many other concepts.

OMDoc/MMT also allows expressing that there are different equivalent definitions of a concept: We defined group actions in two ways and used *views* to express their equivalence.

Representation Level Abstract groups are represented in different ways as concrete objects suitable for computation: as groups of permutations, groups of matrices, finitely presented groups, algebraic constructions of groups, or using polycyclic presentations.

Many representations arise naturally from *group actions*: If we are considering symmetry in a setting where we want to apply group theory, we start with a group action, for example a group acting on a graph by permuting its vertices.

The universal tool to bridge the gap between groups, representations and canonical representatives are *group homomorphisms*, particularly embeddings and isomorphisms, which are used extensively in GAP. This is reflected in our approach.

Implementation Level At this level we encode implementation details: Permutation groups in GAP are considered as finite subgroups of the group $S_{\mathbb{N}+}$, and defined by providing a set of generating permutations. GAP then computes a stabiliser chain for a group that was defined this way, and naturally considers the group to be a subgroup of $S_{[1..n]}$, where n is the largest point moved.

Concrete Level It is at the concrete level where the computation happens: while the higher levels are suitable for mathematical deduction and inference, this level is where GAP (or another system providing computational group theory) does its work.

If a group (or a group action) has been constructed by giving generators through MitM, GAP can now compute the size of the group, its isomorphism type, and perform all the other operations that are available via the GAP system dialect.

4 The System Dialects of GAP, SageMath, and Singular

We now show how we, for GAP, Singular, and SageMath, generate theory graphs that specify the system dialects. The three systems are sufficiently different that we can consider the development presented in this section a meaningful case study in the methodology and investment of exposing the APIs of real-world systems in the form of OMDoc/MMT theories.

In each case, we had to overcome major implementation differences and invest significant manpower. In fact, even the serialization of internal abstract syntax trees as OMDoc/MMT objects proved very difficult, for different system-specific reasons. In the following, we summarize these efforts.

4.1 SageMath

We first consider some related work [Deh+16] regarding a direct (i.e., without MitM) integration of SageMath and GAP. Here SageMath's native interface to GAP is upgraded from the *handle* paradigm to the *semantic handles* paradigm. In the former, 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 (unless it is of some basic type); 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. Additionally, with *semantic* handle, h behaves in A as if it was a native A object. In other words, one adapts 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 SageMath handle `h` to a GAP object `G` should trigger in GAP the corresponding function call `Size(G)`.

This approach avoids the overhead of back and forth conversions between A and B and enables the manipulation from A of objects of B, even if they have no native representation in A. However, in Jane's scenario, we actually want to convert the objects r between A and B.

API In [Deh+16] we describe the extraction of some of SageMath's API from its *categories*. This exploited the mathematical knowledge explicitly embedded in the code to cover a fairly large area of mathematics (hundreds of kinds of algebraic structures such as groups, algebras, fields, ...), with little additional efforts or need to curate the output. This extraction did not cover the *constructors*, knowledge about which is critical for (de)serialization, nor other areas of mathematics (graph theory, elliptic curves, ...) where SageMath developers currently do not use categories (usually because the involved hierarchies of abstract classes are shallow to maintain by hand).

To extract more APIs, we took the following approach:

1. We constructed a list of typical SageMath objects.
2. We used introspection to analyze those objects, crawling recursively through their hierarchy of classes to extract constructors and available methods together with some mathematical knowledge.

At this stage, the list of objects was crafted by hand to cover Jane's scenarios and some others. In a later step we plan to take advantage of one of SageMath's coding standards: every concrete type must be instantiated at least once in SageMath's tests, and the instance be passed trough a generic test suite that runs sanity checks for its advertised properties (e.g. associativity, ...). Therefore, by a simple instrumentation of SageMath's test framework, we could run our exporter on a fairly complete collection of SageMath objects.

The process remains brittle and the export will eventually require much curation:

- The signature of methods is incomplete: it specifies only the number and name of the arguments, and the type of the first argument.
- For constructors, the type of all the arguments is known, but only for the specific call that led to the construction of the introspected object.
- There is no distinction between mathematically relevant methods and purely technical ones like data structure manipulation helpers.
- The export is very large and seems of limited use without alignments with the MitM ontology. At this stage we do not foresee much opportunities to produce such alignments other than manually.

Nonetheless, we consider this an important first step toward fully automatic extraction of the SageMath API. Moreover, we expect further improvements by code annotations in SageMath (e.g., during the ongoing porting of SageMath from Python 2 to Python 3) or using type inference in MitM.

Serialization and Deserialization SageMath being based on Python benefits from its native serialization support. For example, the dihedral group D_4 is serialized as a binary string which encodes the following straight line program to be executed upon deserialization:

```
pg_unreduce = unpickle_global('sage.structure.unique_representation', 'unreduce')
pg_DihedralGroup =
    unpickle_global('sage.groups.perm_gps.permgroup_named', 'DihedralGroup')
pg_make_integer = unpickle_global('sage.rings.integer', 'make_integer')

pg_unreduce(pg_DihedralGroup, (pg_make_integer('4'),), {})
```

The first three lines recover the constructors for integers and for dihedral groups from SageMath's library. The last line applies them to construct successively the integer 4 and D_4.

Up to syntax, this serialization is close to the desired SageMath's OpenMath dialect. We are therefore planning to extend Python's native (de)serializer to use OMDoc/MMT as an alternative serialization format (using the Python library [POMa]).

Here is some motivation for this approach:

- Python's serialization natively implements a certain number of natural optimizations, like factorization of identical subexpressions.
- This decouples nicely the problem into two independent tasks:

1. Instrument the serialization to generate OpenMath as alternative format.
2. Improve the serialization of SageMath objects to serialize *by construction* rather than *by data structure* to hide implementation details and make for straightforward alignments.

– Good serialization support is a long time community goal for SageMath, in order to enable communication between parallel processes, build databases, etc. In addition serialization by construction is already valued as superior, being usually more concise and more resistant to changes across versions of SageMath. Therefore we can hope for long term progress there even if 2. is a large task and there is little man power specifically dedicated to OpenMath support.

4.2 GAP

In [Deh+16], we already described our general approach to extract APIs from the GAP system. We have now improved on this work considerably.

Firstly, we improved the MitM foundation so that the primitives of GAP's type system can be expressed in the MitM ontology.[2] GAP's type system is based on sub-typing: *filters* express finer and finer subtypes of the universal type IsObject. An object in GAP can learn about its properties, meaning its type is refined at runtime: a group can learn that it is Abelian or nilpotent and change its type accordingly.

Secondly, devised and implemented a special treatment of GAP's constructors during serialization. As GAP only has a weak notion of object construction, we achieved this by manually identifying and annotating all functions that create objects in the GAP code base and then instrumenting them to store which arguments they were called with. With the constructor annotation in place, it is possible to have GAP represent any object in a running session as either a primitive type (integers, permutations, transformations, lists, floats, strings), or as a constructor applied to a list of arguments.

The instrumentation itself is minimal – 57 lines of GAP code, plus 100 lines for serializing and parsing. The main – and indeed considerable – challenge was to identify the constructors and their arguments. In GAP, objects are created by calling the function Objectify with a type and some data, hence we analyzed all call-sites to this function and some light inference of the enclosing function. This amounted to 665 call sites in the GAP library and an additional 1664 in the standard package distribution. The instrumentation will be integrated into a future version of GAP, making GAP fully MitM capable.

As a major positive side-effect of our work, this instrumentation led to general improvements of the type infrastructure in GAP. For example, it enables static type analysis, which can be used to optimize the dynamic method dispatch and thus hopefully lead to efficiency gains in the system.

[2] We are hopeful that in the future MMT can even serve as an external type-checker for GAP.

4.3 Singular

As we only need a very small part of Singular for our case study, we were able to use the existing OpenMath content dictionaries for polynomials [OMCP]. These are part of a standard group of content dictionaries that describe (some) mathematical objects at a high level of abstraction to be universally applicable. OMDoc/MMT understands OpenMath, i.e., it can use these content dictionaries as OMDoc/MMT theories.

Building on the OpenMath toolkits for OpenMath phrasebooks [POMa] and SCSCP communication [POMb] in Python – which were developed for SageMath in the OpenDreamKit project, we wrapped Singular in a thin layer of Python code that provides SCSCP communication. This work was undertaken by the sixth author as part of a summer internship in about a week without prior expert knowledge of the system.

4.4 Alignments

Finally we have to curate the alignments between the system dialects and the MitM ontology. These alignments were produced manually.

In the future, we will also consider automatically extracting alignments from the existing ad-hoc SageMath-to-X translations. These are (mainly) given as SageMath code annotations that relate SageMath operations and constructors with those of system X.

5 Distributed Computational Group Theory

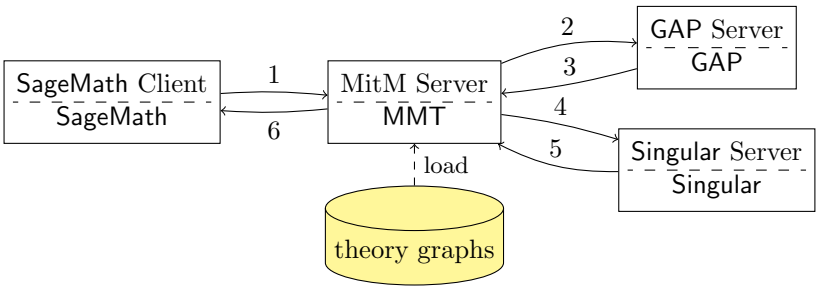

Fig. 4. MitM Interaction in Jane's Use Case

Figure 4 shows the overall architecture with an MitM server as the central mediator. All arrows represent the transfer of OMDoc/MMT ojbects via SCSCP. Critically, the MitM server also implements alignments and uses them to convert between system dialects.

We have extended the MMT system [Rab13] with an SCSCP server/client so that it can receive objects from computation systems and generates calls to others. For the GAP server, we built on pre-existing SCSCP support. To obtain an SCSCP server for Singular, which does not have native SCSCP support, we wrapped Singular in a python script that includes the pyscscp library [POMb]. In SageMath, we directly programmed the client interface to the MitM server.

The numbers on the edges indicate the order of communications when processing Jane's use case. Initially, Jane has already built in SageMath the ring $R = \mathbb{Z}[X_1, X_2, X_3, X_4]$, the group $G = D_4$, and the action A of G on R that permutes the variables, and the polynomial $q = 3 * X_1 + 2 * X_2$. She now calls MitM.Singular(MitM.Gap.orbit(G, A, q)).Ideal().Groebner().sage(), which results in the following steps:

1. Jane uses SageMath to call the MitM server with the respective SageMath object, along with metadata about which system should be used for which computation.
2. The MitM server translates MitM.Gap.orbit(G, A, q) to the GAP system dialect and sends it to GAP.
3. GAP returns the orbit O.
4. The MitM server translates MitM.Singular(O).Ideal().Groebner() to the Singular system dialect and sends it to Singular.
5. Singular returns the Gröbner base B.
6. The MitM server translates B to the SageMath system dialect and sends it to SageMath, where the result is shown to Jane.

6 Conclusion

We have implemented the MitM approach to integrating mathematical software system based on formalizations of the underlying mathematical knowledge. The main investment here was the curation of an MitM Ontology, the generation of formal specifications of system APIs for SageMath, GAP, and Singular, identifying the alignments of these APIs with the ontology, implementing an MitM server that can use alignments to translate between systems, and implementing the SCSCP protocol for all involved systems.

Our case study showed that the MitM-based integration is an achievable goal. Delegation-based workflows can either be programmed directly or embedded into the interaction language of the mathematical software systems.

The main advantages and challenges claimed by the MitM framework come from its loosely coupled, knowledge-based nature. Compared to ad-hoc translations, MitM-based interoperability is relatively expensive as objects have to be serialized into (possibly large) OMDoc/MMT objects, transferred via SCSCP to MMT, parsed, translated into another system dialect, serialized and transferred, and parsed again. On the other hand, instead of implementing and maintaining n^2 translations, we only have to establish and maintain n collections of system APIs and their alignments to the MitM ontology. This makes the management of interoperability much more tractable:

1. The MitM ontology is developed and maintained as a shared resource by the community. We expect it to be well-maintained, since it can directly be used as a documentation of the functionality of the respective systems.
2. All the workflows are star-shaped: instead of requiring expert knowledge in two systems – a rare commodity even in open-source projects, and even for the system experts involved in this paper – and keeping up with their changes, the MitM approach only needs expertise and change management for single systems.

All in all, these translate into a "business model" for the general MitM-based cooperation mode in terms of the necessary investment and achievable results, which is based on the well-known *network effects*: the joining costs are in the size of the respective system, whereas the rewards – i.e. the functionality available by delegation – is in the size of the network.

This network effect can be enhanced by technical refinements we are currently studying: For instance, if we annotate alignments with a "priority" value that specifies how canonical/efficient/powerful a given system is for a given MitM concept, then we can let the MMT mediator automatically choose a suitable target system for a requested computation (as opposed to our current setup where Jane specifies which systems she wants to use). On the other hand, for workflows where we do not need or want service-discovery, alignments can be "compiled" into n^2 transport-efficient direct translations that may even eliminate the need for serialization and parsing.

Acknowledgements The authors gratefully acknowledge the fruitful discussions with other participants of work package WP6, in particular Alexander Konovalov on SCSCP, Paul Dehaye on the SageMath export and the organization of the MitM ontology, and Luca de Feo on OpenMath phrasebooks and a SCSCP library in python. We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

References

[Cre16] John Cremona. "The L-Functions and Modular Forms Database Project". In: *Foundations of Computational Mathematics* 16.6 (2016), pp. 1541–1553. DOI: 10.1007/s10208-016-9306-z.

[Deh+16] Paul-Olivier Dehaye et al. "Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach". In: *Intelligent Computer Mathematics 2016*. Ed. by Michael Kohlhase et al. LNAI 9791. Springer, 2016. URL: https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf.

[Fre+] Sebastian Freundt et al. *Symbolic Computation Software Composability Protocol (SCSCP)*. Version 1.3. URL: https://github.com/OpenMath/scscp/blob/master/revisions/SCSCP_1_3.pdf (visited on 08/27/2017).

[GAP] The GAP Group. *GAP – Groups, Algorithms, and Programming*. URL: http://www.gap-system.org (visited on 08/30/2016).

[Gon+13] Georges Gonthier et al. "A Machine-Checked Proof of the Odd Order Theorem". In: *Interactive Theorem Proving*. Ed. by Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie. Vol. 7998. LNCS. Springer, 2013, pp. 163–179. DOI: 10.1007/978-3-642-39634-2_14.

[Jup] *Project Jupyter*. URL: http://www.jupyter.org (visited on 08/22/2017).

[Koh+11] Michael Kohlhase et al. "The Planetary System: Web 3.0 & Active Documents for STEM". In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*. Ed. by Mitsuhisa Sato et al. Finalist at the Executable Paper Grand Challenge, pp. 598–607. DOI: 10.1016/j.procs.2011.04.063.

[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.

[LMFDB] The LMFDB Collaboration. *The L-functions and Modular Forms Database*. URL: http://www.lmfdb.org (visited on 02/01/2016).

[Mita] *MitM/Foundation*. URL: https://gl.mathhub.info/MitM/Foundation (visited on 09/01/2017).

[Mitb] *MitM/groups*. URL: https://gl.mathhub.info/MitM/groups (visited on 09/01/2017).

[MMT] *MMT – Language and System for the Uniform Representation of Knowledge*. project web site. URL: https://uniformal.github.io/ (visited on 08/30/2016).

[Mül+17a] Dennis Müller et al. "Alignment-based Translations Across Formal Systems Using Interface Theories". In: *Fifth Workshop on Proof eXchange for Theorem Proving - PxTP 2017*. 2017. URL: http://jazzpirate.com/Math/AlignmentTranslation.pdf.

[Mül+17b] Dennis Müller et al. "Classification of Alignments between Concepts of Formal Mathematical Systems". In: *Intelligent Computer Mathematics (CICM) 2017*. LNAI. in press. Springer, 2017. URL: http://kwarc.info/kohlhase/papers/cicm17-alignments.pdf.

[OMCP] *OpenMath CD Group: polygrp*. URL: http://www.openmath.org/cdgroups/polygrp.html (visited on 09/01/2017).

[POMa] *An OpenMath 2.0 implementation in Python*. URL: https://github.com/OpenMath/py-openmath (visited on 09/04/2016).

[POMb] *An SCSCP module for Python*. URL: https://github.com/OpenMath/py-scscp (visited on 09/04/2016).

[Rab13] Florian Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013, pp. 339–343. DOI: 10.1007/978-3-642-39320-4.

[RK13] Florian Rabe and Michael Kohlhase. “A Scalable Module System”. In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: http://kwarc.info/frabe/Research/mmt.pdf.

[Sage] The Sage Developers. *SageMath, the Sage Mathematics Software System*. URL: http://www.sagemath.org (visited on 09/30/2016).

[SNG] *Singular*. URL: https://www.singular.uni-kl.de/ (visited on 08/22/2017).