Trustable Communication Between Mathematics Systems*

Jacques Carette, William M. Farmer, and Jérémie Wajs

Department of Computing and Software
McMaster University
1280 Main Street West
Hamilton, Ontario L8S 4K1
Canada

{carette,wmfarmer,wajs}@mcmaster.ca

Abstract. This paper presents a rigorous, unified framework for facilitating communication between mathematics systems. A mathematics system is given one or more interfaces which offer deductive and computational services to other mathematics systems. To achieve communication between systems, a client interface is linked to a server interface by an asymmetric connection consisting of a pair of translations. Answers to requests are trustable in the sense that they are correct provided a small set of prescribed conditions are satisfied. The framework is robust with respect to interface extension and can process requests for abstract services, where the server interface is not fully specified.

Keywords: Mechanized mathematics, computer theorem proving, computer algebra, intersystem communication, knowledge representation.

1 Introduction

Current mechanized mathematics systems (MMSs), by and large, fall into one of three camps: numerics-based (like Matlab, Octave, and Scilab), symbolic (Maple, Mathematica, MuPAD, and a whole host of smaller systems), and theorem provers (Coq, HOL, IMPS, Isabelle, Nqthm, Nuprl, Otter, and PVS, to name just a few). Each has its strong points, although many are more often bemoaned for their weaknesses. These weaknesses are frequently all the more frustrating for users of many of these systems, as one system's weakness is another's strength. An increasing majority of users are growing to be agnostic in their choice of MMS, worrying more about getting a particular task done than whether one übersystem can do it all. Furthermore, it is important to remark that the expertise needed to build each kind of system is markedly different for all three flavors. Although there has been some efforts at making some of these MMSs broader, familiarity with them quickly dispels any notion that

* This research was supported by Bell Canada and MITACS.

this dabbling is particularly successful. It would perhaps be wiser for the builders of these systems to stay within their sphere of expertise, and enable their software to instead request the services of other systems that can better perform these tasks.

In simple terms, the problem we wish to address, illustrated in Figure 1, is the following: if system A needs access to a certain functionality f which it does not currently implement, but a service providing this functionality is offered by system B, then A should be able to send a request to B containing a translation of its exact problem into the language of B, wait for B to perform the service, and then finally receive an answer in its own language.

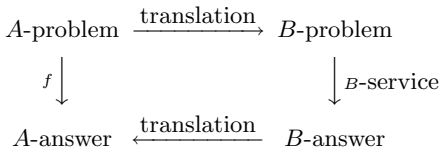

Fig. 1. The basic communication problem

Informally, we wish to think of "perform f" as a request, the pair of translations above as a connection, and the set of available functions from B-problems to B-answers as B's services. We then want to assert that *meaningful communication* happens when the diagram above commutes.

In this paper we present a unified framework which clearly defines these various concepts (**interfaces**, **services**, **connections**, **requests**, and **answers**) in precise mathematical terms. The overarching concern is that of *trust*: when one system requests a service from another, can it trust the result it gets back? Certainly any system which purports to be trustable must also insist that any communication it makes to another system satisfy the same requirements.

1.1 Useful Communication

Certainly examples of *useful* communication between systems abound! Commercial system builders are definitely convinced of this fact, as evidenced by Mathematica's J/Link, Maple's Matlab package, Matlab's Symbolic Toolbox, and so on. Research papers in this area are also plentiful, as we will see later.

For example, polynomial arithmetic is frequently a necessary step in a proof; typical theorem provers will, at best, implement this using rewrite rules (or their equivalent), which is at least an order of magnitude slower than any implementation by any Computer Algebra System (CAS) [8]. In

the opposite direction, closed-form integration of even simple expressions containing parameters involves complex algorithms but also complex side conditions which must be verified, forcing a CAS to call a theorem prover (see [1] and the references therein).

There are also other cases where for exact problems it is possible to compute an exact condition number for the associated numerical problem—one can then use arbitrary precision arithmetic as a decision procedure to test zero-equivalence. Achieving this could involve communication between more than two systems.

1.2 Obstacles

The rise in various technologies like TCP/IP, sockets, and XML has provided some convenient solutions for some of the old obstacles to communicating between MMSs. But a much larger obstacle has arisen: that of semantics. Referring back to Figure 1, it should be clear that describing the semantics of each arrow, in all cases and for all possible services, is nontrivial. To achieve our aim of *trustability*, this issue is inescapable.

To a lesser extent, these is also a problem of interpretability: even if the answer makes sense in System A, is it "the" answer? The notion of "the" answer in a theorem proving system is qualitatively different than from a system centered on numerical analysis, even though both are rigorously and uniquely defined.

1.3 What is Needed

To overcome the obstacles listed in the previous section, one is quite naturally led to formalizing each of the components of Figure 1. Informally, given a system X, let us define an *X-problem* as an expression E of X. Given a function $f : E \mapsto E'$ in X, then define an *X-solution* to simply be that expression E'.

First, let us assume that systems A and B are known, and that the arrow *service* is actually the identity. Meaningful communication between A and B must necessarily imply that f is (at least) an equivalence relation, and furthermore the translations between A and B must in some sense be "meaning preserving". Of course, the closer f is to the identity, the more useful the results will be, but something weaker is all we need for meaningful communication to occur. This needs to be made very precise, and will lead to the definition of a *connection*.

Second, let us assume again that A and B are known and there is connection between them. To get at B's services, B must expose some of its functionality, leading to the concept of an *interface*.

Up to now, neither issues of trustability nor interpretability have really been addressed. It is quite possible that the translation from an A-problem to a B-problem is inaccurate, yet the answer returned by the identity service is correct: for example, the A to B translation could swap some symbols (like $+$ and $*$) and the B to A translation swaps them back. To ensure that both the problem and its answer are properly translated, we first demand that types be respected. This helps but is not sufficient. What we really have to do is to completely change the definition of a solution: instead of getting an expression, we ask for a theorem, which encodes both the problem and its solution. We can then insist that a translation from B back to A be much stronger, that in fact it be an *interpretation*. As we will show, this is enough to ensure that we get a trustable, interpretable answer.

To be explicit, we wish to (re)define an *X-solution* as the theorem $f(E) \equiv \tilde{f}(E)$ where $\tilde{f}$ is a syntactic representation of the action of f.

1.4 Previous Proposals

Several attempts at addressing the problem of communication between MMSs have been made. We can classify them into two categories: the first category consists of work that attempts to deal with the problem in general. Armando and Zini's Logic Broker Architecture [2, 3], Bertoli, Calmet, Giunchiglia and Homann's OMSCS [9], Kerber, Kohlhase and Sorge's Ω-MKRP [16], the OpenMath project [11], and OMDoc [17] fit in this category. The new European MOWGLI project [4], which aims at providing a common machine-understandable (semantics-based) representation of mathematical knowledge and a platform to exploit it, likely fits here too.

The OpenMath project claims to provide a common platform for communication between various mathematics systems. However, while it provides a common syntax, it fails in our view to specify a semantics for that syntax, which is a major drawback when trying to make mathematics systems based on different logics communicate. In other words, there are too many implicit assumptions behind OpenMath's version of semantics for it to apply outside the narrow (but useful) realm of standard operations between the standard CASs.

A refinement to the OpenMath approach lies in OMDoc. The OMDoc approach recognizes the need for semantics, and introduces them through their notion of *theories*. However, OMDoc does not seem to address the

actual mechanics of getting different systems to communicate as much as it provides a common language (syntax + semantics) for them to do so.

The Ω-MKRP approach argues that explicit proofs are needed, that "external" systems cannot be trusted. This seems very impractical.

The OMSCS (Open Mechanized Symbolic Computation Systems) work provides an architecture used to formally specify automated theorem provers and CASs and to formally integrate them. However, it does not seem to address the issues of trust or extending theories.

The Logic Broker Architecture attempts to define a general framework for communication between MMSs. This approach is conceptually very similar to ours. It defines interfaces for MMSs and uses a *Logic Broker* (LB) to perform communication between systems. The LB includes facilities for translation of requests and meaning-preserving translation of answers (thus addressing the question of trust), as well as (in theory) a logical specification matcher to match requests to services offered. However, we believe that this architecture does not support extending theories well, which we will show can be achieved effectively by our approach.

The second category of related work consists of *ad hoc* solutions. In many such cases in the literature, only unidirectional cooperation exists: one system acts as a master, generating requests, while the other one serves as a slave, fulfilling those requests. This includes Howe's work on embedding an HOL theory into Nuprl [15], Ballarin and Paulson's work on using the Sumit library for proofs in Isabelle [8, 6], and Ballarin, Homann and Calmet's work on an interface between Isabelle and Maple [7]. Ballarin and Paulson's work clearly identifies the issue of trust, and distinguishes between trustable results, for which a formal proof exists, and ad hoc results, based on approximations.

Another more evolved *ad hoc* case, intended for bidirectional cooperation between MMSs, is Harrison and Théry's work on combining HOL and Maple [14]. Similarly to Ballarin and Paulson, Harrison and Théry classify the systems by *degree of trust*, for example trusting results proved by HOL while checking results given by Maple.

All these *ad hoc* solutions have the major drawback of not seeking generality. Howe, for instance, does not attempt to make HOL and Nuprl *communicate* as much as he attempts to *embed* an HOL theory into Nuprl. Why should the machinery for HOL be duplicated in Nuprl when it already exists in HOL itself? In addition, this approach is not valid when the system to be integrated is a black box. Our approach enables one MMS to use another MMS's services without, first, having to reproduce them, and second, having to know in detail how they work. We will show how it

addresses the issue of trust, and eliminates the need to verify every single result (which can be painfully burdensome).

1.5 Organization of the Paper

In section 2, we give definitions for the underlying theory necessary to the presentation of our framework. In section 3, we give a simple framework for communication between MMSs. In section 4, we discuss additional obstacles in achieving communication in real cases, and show how to refine the framework presented in section 3 to address some of those obstacles. Finally, we conclude in section 5.

2 Biform Theories and Interpretations

At the heart of this work lies the notion of a "biform theory", which is the basis for FFMM, a Formal Framework for Managing Mathematics [13]. Informally, a biform theory is simultaneously an axiomatic and an algorithmic theory.

In this section we define concepts necessary to understand the rest of this paper. Most of the definitions given here are updated versions of definitions given in [13].

2.1 Logics

A *language* is a set of typed expressions. The types include $*$, which denotes the type of truth values. A *formula* is an expression of type $*$. A *logic* is a set of languages with a notion of logical consequence. If $\mathbf{K}$ is a logic, L is a language of $\mathbf{K}$, and $\Sigma \cup \{A\}$ is a set of formulas of L, then $\Sigma \models_{\mathbf{K}} A$ means that A is a logical consequence of Σ in $\mathbf{K}$.

2.2 Transformers and Formuloids

Let L_i be a language for $i = 1, 2$. A *transformer* Π from L_1 to L_2 is an algorithm that implements a partial function $\pi : L_1 \rightharpoonup L_2$. For $E \in L_1$, let $\Pi(E)$ mean $\pi(E)$, and let $\mathsf{dom}(\Pi)$ denote the domain of π, i.e., the subset of L_1 on which π is defined. For more on transformers, see [12, 13].

A *formuloid* of a language L is a pair $\theta = (\Pi, M)$ where:

1. Π is a transformer from L to L.
2. M is a function that maps each $E \in \mathsf{dom}(\Pi)$ to a formula of L.

M is intended to give the *meaning* of applying Π to an expression E. $M(E)$ usually relates the input E to the output $\Pi(E)$ in some way; for many transformers, $M(E)$ is the equation $E = \Pi(E)$, which says that Π transforms E into an expression with the same value as E itself.

The *span* of θ, written $\mathsf{span}(\theta)$, is the set $\{M(E) \mid E \in \mathsf{dom}(\Pi)\}$ of formulas of L. Thus a formuloid has both an *axiomatic meaning*—its span—and an *algorithmic meaning*—its transformer. The purpose of its span is to assert the truth of a set of formulas, while its transformer is meant to be a deduction or computation rule.

2.3 Biform Theories

A *biform theory* is a tuple $T = (\mathbf{K}, L, \Gamma)$ where:

1. $\mathbf{K}$ is a logic called the *logic* of T.
2. L is a language of $\mathbf{K}$ called the *language* of T.
3. Γ is a set of formuloids of L called the *axiomoids* of T.

The *span* of T, written $\mathsf{span}(T)$, is the union of the spans of the axiomoids of T, i.e., $\bigcup_{\theta \in \Gamma} \mathsf{span}(\theta)$. A is an *axiom* of T if $A \in \mathsf{span}(T)$. A is a *theorem* of T, written $T \models A$, if $\mathsf{span}(T) \models_{\mathbf{K}} A$. A *theoremoid* of T is a formuloid θ of L such that, for each $A \in \mathsf{span}(\theta)$, $T \models A$. Obviously, each axiomoid of T is also a theoremoid of T. An axiomoid is a generalization of an axiom; an individual axiom A can be represented by any axiomoid (Π, M) such that $\mathsf{dom}(\Pi) = \{E\}$ and $M(E) = A$.

T can be viewed as simultaneously both an *axiomatic theory* and an *algorithmic theory*. The axiomatic theory is represented by

$$T_{\text{axm}} = (\mathbf{K}, L, \{M \mid (\Pi, M) \in \Gamma \text{ for some } \Pi\}),$$

and the algorithmic theory is represented by

$$T_{\text{alg}} = (\mathbf{K}, L, \{\Pi \mid (\Pi, M) \in \Gamma \text{ for some } M\}).$$

Let $T_i = (\mathbf{K}, L_i, \Gamma_i)$ be a biform theory for $i = 1, 2$. T_2 is an *extension* of T_1, written $T_1 \leq T_2$, if $L_1 \subseteq L_2$ and $\Gamma_1 \subseteq \Gamma_2$. T_2 is a *conservative extension* of T_1, written $T_1 \trianglelefteq T_2$, if $T_1 \leq T_2$ and, for all formulas A of L_1, if $T_2 \models A$, then $T_1 \models A$. Note that $\leq$ and $\trianglelefteq$ are partial orders.

2.4 Translations and Interpretations

Let $\mathbf{K}_i$ be a logic and $T_i = (\mathbf{K}_i, L_i, \Gamma_i)$ be a biform theory for $i = 1, 2$. A *translation* from T_1 to T_2 is a transformer Φ from L_1 to L_2 that respects

types, i.e., if E_1 and E_2 are expressions in L_1 of the same type and $\Phi(E_1)$ and $\Phi(E_2)$ are defined, then $\Phi(E_1)$ and $\Phi(E_2)$ are also of the same type. T_1 and T_2 are called the *source theory* and the *target theory* of Φ, respectively. Φ is *total* if $\Phi(E)$ is defined for each $E \in L_1$. Φ *fixes* a language L if $\Phi(E) = E$ for each $E \in L$.

An *interpretation* of T_1 in T_2 is a total translation Φ from T_1 to T_2 such that, for all formulas $A \in L_1$, if $T_1 \models A$, then $T_2 \models \Phi(A)$. An interpretation thus maps theorems to theorems. A *retraction* from T_2 to T_1 is an interpretation Φ of T_2 in T_1 such that $T_1 \leq T_2$ and Φ fixes L_1.

Lemma 1. *Let Φ_1 be a retraction from T_2 to T_1 and Φ_2 be a retraction from T_3 to T_2. Then $\Phi_1 \circ \Phi_2$ is a retraction from T_3 to T_1.*

Proof. Let $\Phi = \Phi_1 \circ \Phi_2$. We first need to prove that Φ is an interpretation. Φ is clearly total. Assume $T_3 \models A$. Then $T_2 \models \Phi_2(A)$ since Φ_2 is an interpretation of T_3 in T_2. In turn, $T_1 \models \Phi_1(\Phi_2(A))$, i.e., $T_1 \models A$ since Φ_1 is an interpretation of T_2 in T_1. Hence, Φ is an interpretation of T_3 in T_1.

By transitivity of $\leq$, since $T_1 \leq T_2$ and $T_2 \leq T_3$, $T_1 \leq T_3$.

Finally, we need to prove that Φ fixes L_1. Let $E \in L_1 \subseteq L_2 \subseteq L_3$. $\Phi_2(E) = E$ since Φ_2 is a retraction from T_3 to T_2 and $E \in L_2$. Similarly, $\Phi_1(\Phi_2(E)) = \Phi_1(E) = E$ since Φ_1 is a retraction from T_2 to T_1 and $E \in L_1$. Hence $\Phi(E) = E$ and Φ fixes L_1. □

Proposition 1. *If Φ is a retraction from T_2 to T_1, then $T_1 \trianglelefteq T_2$.*

Proof. Let A be a formula of the language of T_1 such that $T_2 \models A$. We must show that $T_1 \models A$. By definition, (1) Φ is a interpretation of T_2 in T_1 and (2) Φ fixes the language of T_1. (1) implies that $T_1 \models \Phi(A)$, and (2) implies $\Phi(A) = A$. Therefore, $T_1 \models A$. □

3 A Simple Communication Framework

We now present a simple communication framework, based on the theoretical notions presented in the previous section, that addresses the problem presented in Figure 1. The framework formalizes the notions we mentioned in the introduction: interface, service, connection, request, and answer. As we will show after this section, the framework does not address some important practical obstacles to effective communication between MMSs. A refined framework, which is more practical and which generalizes the simple framework here, is presented in section 4.

An *interface* is a pair $I = (T, \mathcal{S})$ where:

1. T is a biform theory called the *theory* of I.
2. $\mathcal{S}$ is a set of theoremoids of T called the *services* of I.

As a theoremoid of T, a service of I is a formuloid whose span is a set of theorems of T and whose transformer is a sound deduction or computation rule for T.

Let $I_i = (T_i, \mathcal{S}_i)$ be an interface for $i = 1, 2$. A *connection* from I_1 to I_2 is a pair $C = (\mathsf{export}, \mathsf{import})$ where:

1. export is a translation from T_1 to T_2.
2. import is an interpretation of T_2 in T_1.

I_1 and I_2 are respectively called the *client interface* and the *server interface* of C. export is for transporting problems from T_1 to T_2; it need not be meaning preserving. import is for transporting solutions from T_2 to T_1; it must be meaning preserving.

An *informed request* is a tuple $R = (I_1, I_2, C, E, \theta)$ where:

1. $I_i = (T_i, \mathcal{S}_i)$ is an interface for $i = 1, 2$.
2. $C = (\mathsf{export}, \mathsf{import})$ is a connection from I_1 to I_2.
3. E is an expression of the language of T_1.
4. $\theta = (\Pi, M) \in \mathcal{S}_2$.

The reason to call such a request *informed* is that it explicitly depends not only on the interface I_2 but on the theoremoid θ as well: we assume that I_1 "knows" about θ. We will come back to this point in section 4.

If $A = \mathsf{import}(M(\mathsf{export}(E)))$ is defined, it is the *answer* to R; otherwise the answer to R is undefined. When A is defined, it is a theorem:

Proposition 2. *Let R and A be as above. If A is defined, then $T_1 \models A$.*

Proof. Assume A is defined. Since θ is a theoremoid of T_2, $T_2 \models M(\mathsf{export}(E))$, and then since import is an interpretation of T_2 in T_1, $T_1 \models \mathsf{import}(M(\mathsf{export}(E)))$. □

The basic problem (Figure 1) is now addressed as shown in Figure 2. All that is necessary to perform this type of communication are interfaces for both systems, and a connection between the two interfaces.

This takes care of the question of *trust* (should A believe the answer it receives from B?), so crucial to the general problem at hand. Whether an answer is correct depends on whether a translation is an interpretation and a service is a theoremoid. Thus an answer is trustworthy if the mechanisms for verifying interpretations and theoremoids are trustworthy.

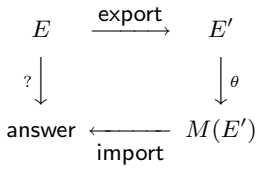

Fig. 2. Communication between two MMSs

Example using Decision Procedures

Suppose S_{hol} is a higher-order interactive theorem proving system with several implemented theories and S_{fol} is a first-order automated theorem proving system with several implemented theories equipped with decision procedures. Suppose further that the underlying logic of S_{hol} is a version of simple type theory and the underlying logic of S_{fol} is a version of first-order logic. This example will show how the framework outlined above can be used to give S_{hol} access to the decision procedures in S_{fol}.

One of the theories of S_{hol} is a theory COF of a complete ordered field which has one model up to isomorphism, namely, the real numbers with the primitive elements 0 and 1 and the primitive operations $+$, $*$, and $<$. An exceedingly rich theory, COF is adequate for developing real analysis. However, several standard decision procedures for theories related to COF are not implemented in S_{hol} but are implemented in S_{fol}.

Two of the theories of S_{fol} are PA, a formalization of first-order Peano arithmetic, and RCF, a formalization of the first-order theory of real closed fields (see [10] for a precise description of these two theories). The theoremoids of PA include θ_+, a decision procedure for additive number theory (Presburger arithmetic), and θ_*, a decision procedure for multiplicative number theory (sometimes called Skolem arithmetic). The theoremoids of RCF include θ_{tarski}, a decision procedure for real closed fields. The framework, with appropriate interfaces and connections, enables these decision procedures to be used within COF.

For example, suppose that we would like to use the decision procedures of PA in COF. Let $I_1 = (\mathsf{COF}, \mathcal{S}_1)$ be an interface of S_{hol} and $I_2 = (\mathsf{PA}, \mathcal{S}_2)$ with $\{\theta_+, \theta_*\} \subseteq \mathcal{S}_2$ be an interface of S_{fol}. Also let $C = (\mathsf{export}, \mathsf{import})$ be the connection from I_1 to I_2 where export translates "first-order natural number formulas" of COF to formulas of PA and import is a standard interpretation of PA in COF (which exists because the natural numbers in COF satisfy Peano's axioms for natural number arithmetic). C offers a way of deciding in COF many statements about the natural numbers using the two decision procedures θ_+ and θ_*, both of which are nontrivial to implement. As an illustration, if the request $R = (I_1, I_2, C, E, \theta_+)$ is made

where E is a formula in COF that expresses the Presburger statement

$$\forall\, a, b, c : \mathbf{N} \;.\; a \equiv b \;\mathsf{mod}\; n \Leftrightarrow a + c \equiv b + c \;\mathsf{mod}\; n,$$

then the answer might be something like $E \Leftrightarrow \mathsf{true}$.

4 A Refined Communication Framework

There are several obstacles to effectively employing the simple framework presented in the previous section. In this section, three obstacles involving connections and one obstacle concerning requests are addressed.

4.1 Obstacles involving Connections

The first obstacle to effectively employing the simple framework is that constructing appropriate connections between interfaces is a challenging task, especially when the biform theories of the interfaces are based on different logics. The export translation of a connection must satisfy a syntactic condition, but the import interpretation must satisfy both a syntactic and semantic condition. As a general principle, it is easier to construct a translation or interpretation Φ if the "primitive basis" of its source theory T_1 (i.e., the primitive symbols and axiomoids of T_1) is small.

The second obstacle is that translating an expression E using the export translation or the import interpretation of a connection may result in an expression much larger than E. As a general principle, it is easier to construct a translation or interpretation Φ without this kind of size explosion if its target theory T_2 contains a rich set of defined symbols.

The third obstacle is that the theory S of an MMS behind the biform theory T of an interface is likely to be enriched with defined symbols over time. Defining a symbol in S will have the effect of extending T to a new theory T'. However, an interpretation Φ of T will not be an interpretation of T' because Φ will not be defined on expressions of T' containing the new defined symbol. As a result, any connection from an interface of the form $(T, \mathcal{S})$ will be broken by the definition of the new symbol.

4.2 Conservative Stacks

The three obstacles described above can be addressed by using a "conservative stack" in place of a biform theory in the definition of an interface. The definitions of interface, connection, informed request, and answer are then redefined. The resulting refined framework is a generalized version of the simple framework.

A *conservative stack* is a pair of sequences $\Sigma = (\tau, \rho)$ where:

1. $\tau = \langle T_0, \ldots, T_n \rangle$ is a finite sequence of biform theories such that, for all i with $0 \leq i < n$, $T_i \leq T_{i+1}$. T_n is called the *theory* of Σ.
2. $\rho = \langle \Phi_1, \ldots, \Phi_n \rangle$ is a finite sequence of translations such that, for all i with $0 < i \leq n$, Φ_i is a retraction from T_i to T_{i-1}.

Notice that, by Proposition 1, the sequence ρ of retractions implies that τ is a "stack" of conservative extensions, i.e., $T_0 \trianglelefteq \cdots \trianglelefteq T_n$.

An *interface* is a pair $I = (\Sigma, \mathcal{S})$ where Σ is a conservative stack and $\mathcal{S}$ is a set of theoremoids of the theory of Σ called the *services* of I.

Let $I_i = ((\tau_i, \rho_i), \mathcal{S}_i)$ be an interface with $\tau_i = \langle T_0^i, \ldots, T_{n_i}^i \rangle$ for $i = 1, 2$. A *connection* C from I_1 to I_2 is a tuple $(T^1, \mathsf{export}, U^2, T^2, \mathsf{import}, U^1)$ where:

1. T^1 and U^1 are members of τ_1.
2. T^2 and U^2 are members of τ_2
3. export is a translation from T^1 to U^2.
4. import is an interpretation of T^2 in U^1.

Let Φ_i be the composition of elements of ρ_i from $T_{n_i}^i$ to T^i for $i = 1, 2$. By Lemma 1, Φ_i is a retraction from $T_{n_i}^i$ to T^i for $i = 1, 2$.

Notice that $(\mathsf{export} \circ \Phi_1, \mathsf{import} \circ \Phi_2)$ is a connection from $(T_{n_1}^1, \mathcal{S}_1)$ to $(T_{n_2}^2, \mathcal{S}_2)$ in the simple framework and that is not necessary that $T_1 = U_1$ or $T_2 = U_2$.

An *informed request* is a tuple $R = (I_1, I_2, C, E, \theta)$ where:

1. I_i is an interface for $i = 1, 2$ as defined above.
2. C is a connection from I_1 to I_2 as defined above.
3. E is an expression of the language of $T_{n_1}^1$, the theory of I_1.
4. $\theta = (\Pi, M) \in \mathcal{S}_2$.

If $A = \mathsf{import}(\Phi_2(M(\mathsf{export}(\Phi_1(E)))))$ is defined (where Φ_1 and Φ_2 are defined as above), it is the *answer* to R; otherwise the answer to R is undefined. When A is defined, it is a theorem:

Proposition 3. *Let R and A be as above. If A is defined, then $U^1 \models A$.*

Proof. Assume that A is defined. Since θ is a theoremoid of $T_{n_2}^2$, the theory of I_2, $T_{n_2}^2 \models M(\mathsf{export}(\Phi_1(E)))$, and since Φ_2 is a retraction from $T_{n_2}^2$ to T^2, $T^2 \models \Phi_2(M(\mathsf{export}(\varphi_1(E)))))$. Since import is an interpretation of T^2 in U^1, we conclude that $U^1 \models A$. $\square$

This refined framework addresses all three of the obstacles discussed above. First, the refined framework facilitates the construction of a connection of a translation or interpretation Φ between I_1 and I_2 by allowing

the source theory of Φ to be chosen from the lower part of the conservative stack of I_1. Second, the refined framework facilitates the construction of a translation or interpretation Φ between I_1 and I_2 without a size explosion by allowing the target theory of Φ to be chosen from the upper part of the conservative stack of I_2. And finally, if a conservative stack Σ is extended to a larger conservative stack Σ', then Σ can be freely replaced with Σ' without compromising any existing interfaces or connections.

4.3 Specifying Requests and Services

In the above framework, we assumed that system A "magically" knows that it wants to use service θ of system B. However, in a more general setting, one would want to specify a request (e.g., *evaluate this computation*), pass on that specification to some entity able to match that request to an available service.

To solve this problem, we need to go up one level: instead of dealing with expressions of L_1, we need to deal with some specification Spec corresponding to some function $f : L_1 \to L_1$ (a computational transformer) associated with a "virtual service" θ_1. Given Spec, the task then becomes one of *finding* an informed request such that the diagram presented in Figure 3 commutes. In theory, this is what we understand the LS Matcher of the Logic Broker [2, 3] of Armando and Zini is somehow supposed to perform, although its task is never defined precisely.

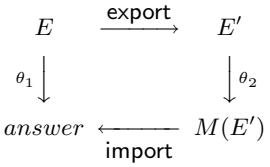

Fig. 3. The specification matching problem

First, we need to shift our focus somewhat. Let us define *reachable services* as those computational theoremoids θ_2 of L_2 that can be given a complete specification in some meta-language Spec. We could, for example, use CASL [5], *Z* [19] or *Specware* [18] for this task. In other words, we would like to define services (and requests) implicitly, allowing nonconstructive definitions as well. Note that we specifically exclude those theoremoids that cannot be finitely axiomatized in some meta-language. Symmetrically to reachable services, we define (brokered) *requests* as those virtual services θ_1 of L_1 which can be specified completely in Spec.

We then need to solve the specification matching problem: given a pair (S_1, S_2) of specifications for θ_1 and θ_2, does there exist a connection C such that Figure 3 commutes.

Even in the simplest possible case where both systems are the same, and thus import and export can be taken to be the identity, this problem is still quite difficult, unless great pains are taken to specify each system's services in a very uniform manner. However, the situation is far from hopeless: even though there are many different ways to specify that, for example, a particular function is a primality verification function (or an implementation thereof), the task of deciding that two such specifications are equivalent is considerably simpler than actually providing a provably correct implementation!

5 Conclusion

In this paper we have presented a mathematically rigorous framework for communicating mathematics between MMSs. This framework gives precise meanings to notions such as *(biform) theories*, *interfaces*, *services*, *connections*, *requests*, and *answers*. It addresses the issue of trust, which has been identified as a central issue in intersystem communication in related papers, by using *interpretations* (meaning-preserving translations) to communicate answers. It also provides facilities for conservatively extending theories, allowing them to evolve as needed without needing to rebuild whole new interfaces or to drastically update connections.

We have not discussed in this paper how biform theories are obtained. For this, we refer the reader to [13].

We have defined precisely the problem of specification of services, and of logical specification matching. We are aware that any useful implementation of the ideas detailed in this paper would need to include such a facility, and we are working in that direction.

References

1. A. Adams, M. Dunstan, H. Gottliebsen, T. Kelsey, U. Martin, and S. Owre. Computer algebra meets automated theorem proving: Integrating Maple and PVS. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2001*, volume 2152 of *LNCS*, pages 27–42. Springer-Verlag, 2001.
2. A. Armando and D. Zini. Towards interoperable mechanized reasoning systems: the logic broker architecture. In A. Corradi, A. Omicini, and A. Poggi, editors, *WOA 2000 — Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, volume 1195 of *Atti di Congressi*, pages 70–75. Pitagora Editrice Bologna, 2000.

3. A. Armando and D. Zini. Interfacing computer algebra and deduction systems via the logic broker architecture. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning, Proceedings of the Eight Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (CALCULEMUS-2000)*, pages 49–64. A.K. Peters, 2001.
4. A. Asperti and B. Wegner. MOWGLI — a new approach for the content description in digital documents. In *Proceedings of the Nineth International Conference on Electronic Resources and the Social Role of Libraries in the Future*, 2002.
5. E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P. D. Mosses, D. Sannella, and A. Tarlecki. CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, 2002.
6. C. Ballarin. *Computer Algebra and Theorem Proving.* PhD thesis, Cambridge University, 1999.
7. C. Ballarin, K. Homann, and J. Calmet. Theorems and algorithms: An interface between Isabelle and Maple. In *International Symposium on Symbolic and Algebraic Computation*, pages 150–157, 1995.
8. C. Ballarin and L. C. Paulson. A pragmatic approach to extending provers by computer algebra - with applications to coding theory. *Fundamenta Informaticae*, 39(1–2):1–20, 1999.
9. P. G. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. In *Fourth International Conference On Artificial Intelligence and Symbolic Computation (AISC'98)*, volume 1476 of *LNCS*. Springer-Verlag, 1998.
10. C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland, 1990.
11. S. Dalmas, M. Gaëtano, and S. M. Watt. An OpenMath 1.0 implementation. In *Proceedings of ISSAC-97*, pages 241–248, 1997.
12. W. M. Farmer and M. v. Mohrenschildt. Transformers for symbolic computation and formal deduction. In S. Colton, U. Martin, and V. Sorge, editors, *Proceedings of the Workshop on the Role of Automated Deduction in Mathematics, CADE-17*, pages 36–45, 2000.
13. W. M. Farmer and M. v. Mohrenschildt. An overview of a formal framework for managing mathematics. *Annals of Mathematics and Artificial Intelligence*, 2003. In the forthcoming special issue: B. Buchberger, G. Gonnet, and M. Hazewinkel, eds., *Mathematical Knowledge Management.*
14. J. Harrison and L. Théry. A skeptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21(3):279–294, 1998.
15. D. J. Howe. Importing mathematics from HOL into Nuprl. In J. Von Wright, J. Grundy, and J. Harrison, editors, *Ninth International Conference on Theorem Proving in Higher Order Logics TPHOL*, volume 1125 of *LNCS*, pages 267–282. Springer-Verlag, 1996.
16. M. Kerber, M. Kohlhase, and V. Sorge. An integration of mechanised reasoning and computer algebra that respects explicit proofs. Technical Report CSRP-96-9, University of Birmingham, 1996.
17. M. Kohlhase. OMDoc: An open markup format for mathematical documents (version 1.1). Technical report, Carnegie Mellon University, 2002.
18. Y. V. Srinivas and R. Jullig. Specware: Formal support for composing software. In *Mathematics of Program Construction*, pages 399–422, 1995.
19. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Series in Computer Science. Prentice Hall, 1996.