

# Trustable Communication Between Mathematics Systems\*

Jacques Carette, William M. Farmer, and Jérémie Wajs\*\*

McMaster University  
Hamilton, Ontario, Canada

**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, Scilab, etc), symbolic (Maple, Mathematica, MuPAD, etc), and theorem provers (Coq, HOL, IMPs, Isabelle, Nqthm, Nuprl, Otter, PVS, etc). Each has its strong points, although many are more often bemoaned for their weaknesses. These weaknesses are all the more frustrating for users as one system's weakness is frequently another's strength. An increasing majority of users are becoming agnostic in their choice of MMSs, 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 have been some efforts at making some of these MMSs broader, familiarity with them quickly dispels any notion that this dabbling is particularly successful. A wiser approach, at least in the medium term, is to construct a larger system out of trusted specialized pieces.

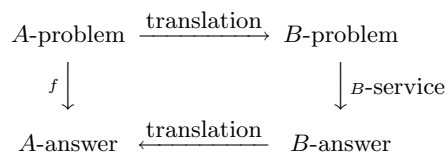
In simple terms, the problem we wish to address, illustrated in Figure 1, is the following: if system  $A$  needs access to a certain function-

---

\* This research was supported by Bell Canada and MITACS.

\*\* {cchette,wmfarmer,wajs}@mcmaster.ca

ality  $f$  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 satisfies the same requirements. We have not generally addressed the concept of *usefulness* of the resulting communication, as we are not aware of any generally accepted mathematical definition of that concept.

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.

For example, polynomial arithmetic is frequently a necessary step in a proof; typical theorem provers will, at best, implement this using rewrite rules, which are at least an order of magnitude slower than implementations by Computer Algebra Systems (CASs) [7]. 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).

We consider old obstacles (issues of transport and syntax) to be essentially solved by common technologies (TCP/IP, sockets, XML, etc). What remains to be solved adequately is the problem of semantics. Referring back to Figure 1, it should be clear that describing 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, there 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 in a system centered on numerical analysis, even though both are rigorously and uniquely defined.

More discussion can be found in the long version [9] of this paper.

The rest of the paper is organized as follows: In section 2, we look at previous related proposals. In section 3, we give definitions for the underlying theory necessary to the presentation of our framework. In section 4, we give a simple framework for communication between MMSs. In section 5, we discuss additional obstacles in achieving communication in real cases, and show how to refine the framework presented in section 4 to address some of those obstacles. In section 6, we talk about specification of requests and services. Finally, we conclude in section 7.

## 2 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. The second category consists of *ad hoc* solutions. We review important members of each category below.

**General Solutions** The OpenMath project [10] 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.

OMDoc [15] constitutes a refinement to the OpenMath approach: it recognizes the need for semantics, and introduces them through a 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. Nevertheless, OMDoc could be extended to handle the concepts of our framework: interfaces, services, connections, requests, and answers.

The  $\Omega$ -MKRP [14] approach argues that explicit proofs are needed and that “external” systems cannot be trusted. This seems very impractical.

The OMSCS (Open Mechanized Symbolic Computation Systems) [8] 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.

Armando and Zini’s Logic Broker Architecture [2], defines 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 achieve 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 new European MOWGLI project [3], which aims at providing a common machine-understandable (semantics-based) representation of mathematical knowledge and a platform to exploit it, likely fits here too.

**Ad-hoc approaches** 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 [13], Ballarin and Paulson’s work on using the Sumit library for proofs in Isabelle [5, 7], and Ballarin, Homann, and Calmet’s work on an interface between Isabelle and Maple [6]. 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 complex *ad hoc* case, intended for bidirectional cooperation, is Harrison and Théry’s work on combining HOL and Maple [12]. Similarly to Ballarin and Paulson, they 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).

### 3 Biform Theories

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 [11]. Informally, a biform theory is simultaneously an axiomatic and an algorithmic theory. Most of the definitions given here are simplified versions of definitions given in [11].

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  $*$ . For a formula  $A$  of a language  $L$ ,  $\neg A$ , the negation of  $A$ , is also a formula of  $L$ . 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  $\Sigma$  in  $\mathbf{K}$ .

Let  $L_i$  be a language for  $i = 1, 2$ . A *transformer*  $\Pi$  from  $L_1$  to  $L_2$  is an algorithm that implements a partial function  $\pi : L_1 \rightarrow L_2$ . For  $E \in L_1$ , let  $\Pi(E)$  mean  $\pi(E)$ , and let  $\text{dom}(\Pi)$  denote the domain of  $\pi$ , i.e., the subset of  $L_1$  on which  $\pi$  is defined.

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

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

$M$  is intended to give the *meaning* of applying  $\Pi$  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  $\Pi$  transforms  $E$  into an expression with the same value as  $E$  itself.

The *span* of  $\theta$ , written  $\text{span}(\theta)$ , is the set  $\{M(E) \mid E \in \text{dom}(\Pi)\}$  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.

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.  $\Gamma$  is a set of formuloids of  $L$  called the *axiomoids* of  $T$ .

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

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

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

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.

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  $\Phi$  from  $L_1$  to  $L_2$  that:

1. 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.
2. Respects negation, i.e., if  $A$  is a formula in  $L_1$  and  $\Phi(A)$  is defined, then  $\Phi(\neg A) = \neg\Phi(A)$ .

$T_1$  and  $T_2$  are called the *source theory* and the *target theory* of  $\Phi$ , respectively.  $\Phi$  is *total* if  $\Phi(E)$  is defined for each  $E \in L_1$ .  $\Phi$  *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  $\Phi$  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. (Since any translation respects negation, an interpretation also maps negated theorems to negated theorems.) A *retraction* from  $T_2$  to  $T_1$  is an interpretation  $\Phi$  of  $T_2$  in  $T_1$  such that  $T_1 \leq T_2$  and  $\Phi$  fixes  $L_1$ .

**Proposition 1.** *If  $\Phi$  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)  $\Phi$  is an interpretation of  $T_2$  in  $T_1$  and (2)  $\Phi$  fixes the language of  $T_1$ . (1) implies that  $T_1 \models \Phi(A)$ , and (2) implies  $\Phi(A) = A$ . Therefore,  $T_1 \models A$ .  $\square$

## 4 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 obstacles to effective communication between MMSs. A refined framework, which is more practical and which generalizes this simple framework, is presented in section 5.

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  $C = (\text{export}, \text{import})$  where **export** is a translation from  $T_1$  to  $T_2$ , and **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** transports solutions from  $T_2$  to  $T_1$ ; it must be meaning preserving.

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

1.  $C = (\text{export}, \text{import})$  is a connection from  $I_1 = (T_1, \mathcal{S}_1)$  to  $I_2 = (T_2, \mathcal{S}_2)$ .
2.  $E$  is an expression of the language of  $T_1$ .
3.  $\theta = (H, M) \in \mathcal{S}_2$ .

The reason to call such a request *informed* is that it explicitly depends not only on the interface  $I_2$  but on the theoremoid  $\theta$  as well: we assume that  $I_1$  “knows” about  $\theta$ . We will come back to this point in section 5.

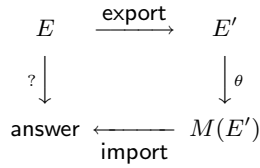
If  $A = (\text{import} \circ M \circ \text{export})(E)$  is defined, it is the *answer* to  $R$ ; otherwise the answer to  $R$  is undefined. When  $A$  is defined, it is a theorem:

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

*Proof.* Assume  $A$  is defined. Since  $\theta$  is a theoremoid of  $T_2$ ,  $T_2 \models (M \circ \text{export})(E)$ , and then since **import** is an interpretation of  $T_2$  in  $T_1$ ,  $T_1 \models (\text{import} \circ M \circ \text{export})(E)$ .  $\square$

Note that, if  $C$  and  $\theta$  are not chosen well,  $A$  may be a useless theorem such as true or  $E = E$ .

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.



**Fig. 2.** Communication between two MMSs

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.

Note also at this point that a given system may have many interfaces, each containing only one or a few services of that system. This approach allows us to consider trustable subsystems within a system and to use those subsystems in trustable communication. For example, while a result given by Maple cannot be fully trusted in general, many subparts of Maple are well encapsulated and could be proved correct.

### Example using Decision Procedures

Suppose  $S_{\text{hol}}$  is a higher-order interactive theorem proving system with several implemented theories including COF, a theory of a complete ordered field. COF has one model up to isomorphism, namely, the real numbers with the usual operations such as  $+$ ,  $*$ , and  $<$ . An exceedingly rich theory, COF is adequate for developing real analysis. Suppose also that  $S_{\text{fol}}$  is a first-order automated theorem proving system with several implemented theories equipped with decision procedures including PA, a theory of first-order Peano arithmetic. The theoremoids of PA include  $\theta_+$ , a decision procedure for additive number theory (Presburger arithmetic), and  $\theta_*$ , a decision procedure for multiplicative number theory (sometimes called Skolem arithmetic). The framework outlined above can be used to give  $S_{\text{hol}}$  access to the decision procedures in  $S_{\text{fol}}$ .

Let  $I_1 = (\text{COF}, \mathcal{S}_1)$  be an interface of  $S_{\text{hol}}$  and  $I_2 = (\text{PA}, \mathcal{S}_2)$  with  $\{\theta_+, \theta_*\} \subseteq \mathcal{S}_2$  be an interface of  $S_{\text{fol}}$ . Also let  $C = (\text{export}, \text{import})$  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. (Because COF satisfies Peano’s (second-order) axioms for natural number arithmetic, (1) **export** is not an interpretation and (2) **import** exists.)  $C$  offers a way of deciding in COF many statements about the natural numbers using the two decision procedures  $\theta_+$  and  $\theta_*$ , both of which are nontrivial to implement. See [9] for further details.



## 5 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 are addressed.

The first obstacle is that constructing 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  $\Phi$  if the “primitive basis” of its source theory  $T_1$  (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  $\Phi$  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  $\Phi$  of  $T$  will not be an interpretation of  $T'$  because  $\Phi$  will not be defined on expressions of  $T'$  containing the new defined symbol. As a result, any connection to an interface of the form  $(T, \mathcal{S})$  will be broken by the definition of the new symbol.

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

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

1.  $\tau = \langle T_0, \dots, T_n \rangle$  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  $\Sigma$ .
2.  $\rho = \langle \Phi_1, \dots, \Phi_n \rangle$  is a finite sequence of translations such that, for all  $i$  with  $0 < i \leq n$ ,  $\Phi_i$  is a retraction from  $T_i$  to  $T_{i-1}$ .

Notice that, by Proposition 1, the sequence  $\rho$  of retractions implies that  $\tau$  is a “stack” of conservative extensions, i.e.,  $T_0 \triangleleft \dots \triangleleft T_n$ .

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

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

1. **export** is a translation from  $U^1$  to  $V^2$ .

2. **import** is an interpretation of  $U^2$  in  $V^1$ .
3.  $U^1$  and  $V^1$  are members of  $\tau_1$ .
4.  $U^2$  and  $V^2$  are members of  $\tau_2$ .

Let  $\Phi_i$  be the composition of elements of  $\rho_i$  from  $T_{n_i}^i$  to  $U^i$  for  $i = 1, 2$ . It is easy to see that  $\Phi_i$  is a retraction from  $T_{n_i}^i$  to  $U^i$  for  $i = 1, 2$ .  $(\text{export} \circ \Phi_1, \text{import} \circ \Phi_2)$  is a connection from  $(T_{n_1}^1, \mathcal{S}_1)$  to  $(T_{n_2}^2, \mathcal{S}_2)$  in the simple framework even if  $U^1 \neq V^1$  or  $U^2 \neq V^2$ .

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

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

If  $A = (\text{import} \circ \Phi_2 \circ M \circ \text{export} \circ \Phi_1)(E)$  is defined (where  $\Phi_1$  and  $\Phi_2$  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  $V^1 \models A$ .*

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

The refined framework facilitates the construction of a translation or interpretation  $\Phi$  between two interfaces  $I_1$  and  $I_2$  by allowing the source theory of  $\Phi$  to be chosen from the lower part of the conservative stack of  $I_1$  and the target theory of  $\Phi$  to be chosen from the upper part of the conservative stack of  $I_2$  (addressing the first and second obstacles discussed above). If a conservative stack  $\Sigma$  is extended to a larger conservative stack  $\Sigma'$ , then  $\Sigma$  can be freely replaced with  $\Sigma'$  without compromising any existing interfaces or connections (addressing the third obstacle).

## 6 Specifying Requests and Services

Until now, we assumed that system  $A$  “magically” knows that it wants to use service  $\theta$  of system  $B$ . However, in a more general setting, one would want to specify a request (like *evaluate this computation*), and pass that specification on to some entity able to match it to an available service.

Thus, instead of dealing with services of  $I_2$ , we need to deal with some specification  $S$  corresponding to some function  $f : L_1 \rightarrow L_1$  (a computational transformer) associated with a “virtual service”  $\theta_1$ . Given

$S$ , the task then becomes one of *finding* an informed request such that our communication diagram commutes. In theory, this is what we understand that Armando and Zini’s LS Matcher [2] is somehow supposed to perform, although its task is never defined precisely.

Let us define *reachable services* as those computational theoremoids  $\theta_2$  of  $T_2$  that can be given a complete specification in some meta-language  $\text{Spec}$ . We could, for example, use CASL [4],  $Z$  [17] or *Specware* [16] for this task. In other words, we wish to define services (and requests) implicitly, allowing nonconstructive definitions as well. Note that we specifically exclude theoremoids that cannot be finitely axiomatized in  $\text{Spec}$ . Symmetrically to reachable services, we define (brokered) *requests* as those virtual services  $\theta_1$  of  $I_1$  which can be specified completely in  $\text{Spec}$ .

We then need to solve the specification matching problem: given a pair  $(S_1, S_2)$  of specifications for  $\theta_1$  and  $\theta_2$ , does there exist a connection  $C$  such that our communication diagram commutes?

Even in the simplest possible case where both systems are the same, this problem can still be 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!

## 7 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 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. Interfacing computer algebra and deduction systems via the logic broker architecture. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning (CALCULEMUS-2000)*, pages 49–64. A. K. Peters, 2001.
3. A. Asperti and B. Wegner. MOWGLI — a new approach for the content description in digital documents. In *Ninth International Conference on Electronic Resources and the Social Role of Libraries in the Future*, Autonomous Republic of Crimea, Ukraine, 2002.
4. 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*, 286:153–196, 2002.
5. C. Ballarin. *Computer Algebra and Theorem Proving*. PhD thesis, Cambridge University, 1999.
6. C. Ballarin, K. Homann, and J. Calmet. Theorems and algorithms: An interface between Isabelle and Maple. In *International Symposium on Symbolic & Algebraic Computation (ISSAC-95)*, pages 150–157, 1995.
7. C. Ballarin and L. C. Paulson. A pragmatic approach to extending provers by computer algebra - with applications to coding theory. *Fundamenta Informaticae*, 39:1–20, 1999.
8. P. G. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. *Fundamenta Informaticae*, 39:39–57, 1999.
9. J. Carette, W. M. Farmer, and J. Wajs. Trustable communication between mathematics systems. Technical report, McMaster University, 2003.
10. S. Dalmas, M. Gaëtano, and S. M. Watt. An OpenMath 1.0 implementation. In *International Symposium on Symbolic & Algebraic Computation (ISSAC-97)*, pages 241–248, 1997.
11. W. M. Farmer and M. v. Mohrenschildt. An overview of a Formal Framework for Managing Mathematics. *Annals of Mathematics and Artificial Intelligence*, 38:165–191, 2003.
12. J. Harrison and L. Théry. A skeptic’s approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.
13. D. J. Howe. Importing mathematics from HOL into Nuprl. In J. Von Wright et al., editors, *Theorem Proving in Higher Order Logics (TPHOLs 1996)*, volume 1125 of *LNCS*, pages 267–282. Springer-Verlag, 1996.
14. 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.
15. M. Kohlhase. OMDoc: An open markup format for mathematical documents (version 1.1). Technical report, Carnegie Mellon University, 2002.
16. Y. V. Srinivas and R. Jullig. Specware: Formal support for composing software. In *Mathematics of Program Construction*, pages 399–422, 1995.
17. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Series in Computer Science. Prentice Hall, 1996.