Communication Protocols for Mathematical Services based on KQML and OMRS

Alessandro Armando[‡], Michael Kohlhase[†], Silvio Ranise[‡]

[†] FR 6.2, Univ. Saarland, Germany
kohlhase@ags.uni-sb.de

[‡] DIST, Università di Genova, Italy
{armando|silvio}@dist.unige.it

Abstract. *In this paper we describe the first ideas for formalizing a communication protocol for mathematical services based on* KQML *(Knowledge Query and Manipulation Language) and* OMRS *(Open Mechanized Reasoning Systems). The claim is that the interaction level of a communication protocol for mathematical services can be relatively generic (hence* KQML *suffices), as long as the ontology of the computational behavior and internal state of the mathematical services is sufficiently expressive and concise (which we have in* OMRS*).*

The material presented in this paper is a first exploratory step towards the definition of the interaction level in OMRS*, supplies a concrete syntax based on the* OPENMATH *standard, and gives a semantics to communication of mathematical services in distributed theorem proving and symbolic computation environments.*

1 Introduction

It is plausible to expect that the way we do (conceive, develop, communicate about, and publish) mathematics will change considerably in the next ten years. The Internet plays an ever-increasing role in our everyday life, and most of the mathematical activities will be supported by mathematical software systems (we will call them *mathematical services*) connected by a commonly accepted distribution architecture, which we will call the *mathematical software bus*. We have argued for the need of such an architecture in [13], and we have in the meantime gained experiences with prototype systems (the MATHWEB software bus [14] (MATHWEB-SB) and the LOGIC BROKER ARCHITECTURE [5] systems); other groups have conducted similar experiments [15, 10] based on other implementation

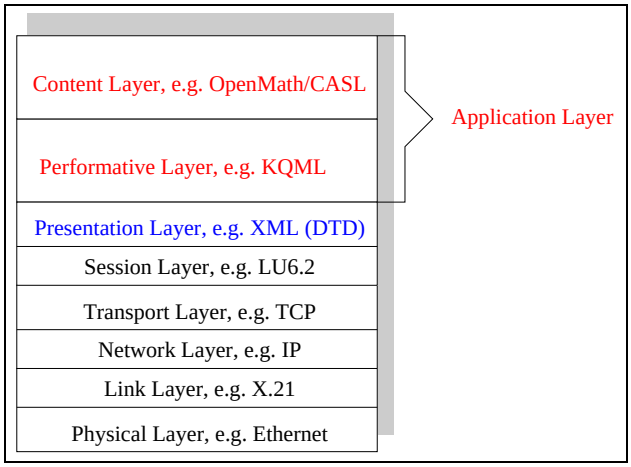


Figure 1. Artificial Communication: KQML and the OSI Reference Model

technologies, but with the same vision of creating a world wide web of cooperating mathematical services. In order to avoid fragmentation, double inventions and to foster ease of access it is necessary to define interface standards for MATHWEB.[1] In [13], we have already proposed a protocol based on the agent communication language KQML [12] and the emerging Internet standard OPENMATH [8, 28] as a content language (see Figure 1). This layered architecture which refines the unspecific "application layer" of the OSI protocol stack is inspired by the results from agent-oriented programming [19, 18], and is based on the intuition, that all agents (not only mathematical services) should understand the agent communication language, even if they do not understand the content language, which is used to transport the actual mathematical content. The agent communication language is used to establish agent identity, reference and—in general—model the communication protocols. The content language is used for transporting the mathematical content.

In this paper, we refine the communication protocol proposed for MATHWEB based on the experiences gained since [13]. The problem with that proposal was that pure OPENMATH [8] is too weak as a content language, since it is geared towards the representation of mathematical objects, which

[1] We will for the purposes of this paper subsume all of the implementations by the term MATHWEB, since the communication protocols presented in this paper will make the constructions of bridges between the particular implementation simple, so that the combined systems appear to the outside as one homogenous web. There is a joint effort underway to do just that at `http://www.mathweb.org`

is sufficient for content communication among symbolic/numeric computation services, but not for reasoning services. In [22] the second author has presented an extension OMDoc (OpenMath Documents) of OpenMath by primitives for document structure, theory management, and proofs, arriving at a (structured) specification language for symbols, definitions, theorems, theories, etc. which make up the content reasoning services need to communicate about. Yet, this is still not sufficient to specify the *interaction* of mathematical services on MathWeb; this is where the Kqml kicks in. In fact, Kqml was developed exactly for the purpose of agent interaction, assuming that the content layer has already been specified.

However, in order to build a protocol based on something as general as the Kqml we need a way to specify the state and the strategies of the mathematical services themselves. This is exactly what the OMRS framework has been developed for: to supply a specification framework for mathematical software systems. In OMRS, a mathematical software system is structured in three layers: the *logic* layer (specifying the deductive machinery), the *control* layer (specifying the strategies for controlling inference), and the *interaction* layer (specifying the interaction capabilities of the mathematical software system with others and with humans).[2] The **main claim** of this paper is that the interaction level of a communication protocol for mathematical services can be relatively generic (hence Kqml suffices), as long as the ontology of the computational behavior and internal state of the mathematical services is sufficiently expressive and concise (which we have in OMRS).

The plan of the paper is as follows. In Section 2, we briefly review Kqml and present a simple ontology for reasoning services in MathWeb (Section 2.1). In Section 3, we hint the main concepts of the OMRS framework. Then, in Section 4, we put together Kqml and OMRS to validate our claim on a significant example. In Section 5 we discuss a migration path for the existing MathWeb software bus implementations towards a joint architecture based on the protocol presented in this paper. Finally, in Section 6, we draw some conclusions.

2 The Kqml and MathWeb

Kqml, the Knowledge Query and Manipulation Language,[3] is a language and protocol for exchanging information and knowledge among software

[2] The OMRS framework—initially conceived to specify deduction systems—has been extended to support the specification of computer algebra systems in [6]. For the sake of simplicity in this paper we use the original OMRS framework.

[3] More information can be obtained from `http://www.cs.umbc.edu/kqml`.

`ask-if`(A,B,X)
Pre: $want(A, know(A, \mathcal{Y}))$
Post: $intend(A, know(A, \mathcal{Y}))$ $know(B, want(A, know(A, \mathcal{Y})))$

`tell`(A,B,X)
Pre: $bel(A, X)$ $know(A, want(B, know(B, \mathcal{Y})))$ $intend(B, know(B, \mathcal{Y}))$
Post: $know(A, know(A, bel(A, X)))$ $know(B, bel(A, X))$

Legend: $\mathcal{Y}$ stands for $bel(B, X)$, $bel(B, \neg X)$, or $\neg bel(B, X)$.

Figure 2. Semantics of KQML performatives

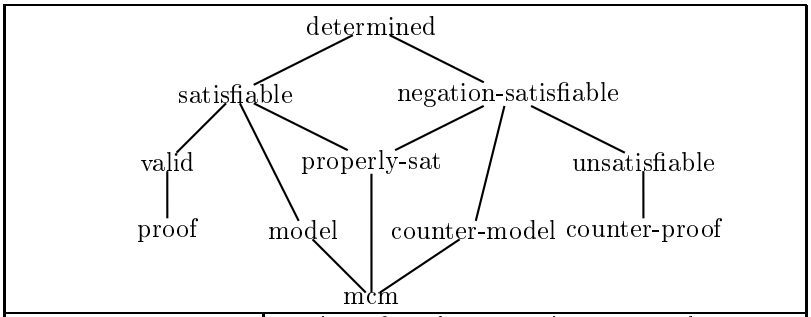
agents. It is both a message format and a message-handling protocol and can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving.

KQML focuses on an extensible set of performatives, which defines the permissible operations that agents may attempt on each other's knowledge and goal stores. The performatives comprise a substrate on which to develop higher-level models of inter-agent interaction such as contract nets and negotiation. Following [24], KQML performatives can be modeled as actions which modify the cognitive states of the agents. The cognitive states of the agents can be modeled by means of the predicates:

- $bel(A, X)$ asserts that X is true for agent A (where X is a statement about the application domain), or equivalently that X is in the (virtual) knowledge base of A.
- $know(A, Y)$ asserts that Y is known to be true by A. (Here and below Y is a statement about the cognitive states of the agents).
- $want(A, Y)$ asserts that A desires the state described by Y to occur.
- $intend(A, Y)$ asserts that A has every intention of achieving the state described by Y.

Notice that while the meaning of *know*, *want*, and *intend* is fixed, the meaning of *bel* depends on the application. The semantics of KQML performatives is given in terms of the preconditions and postconditions describing the applicability conditions and the effect of the performatives respectively. A (simplified) account of the semantics of the `ask-if` and `tell` performatives is given in Figure 2. (The specification of the `deny` performative can be obtained from that of `tell` by replacing every occurrence of $bel(A, X)$ with $\neg bel(A, X)$.)

In addition, KQML provides a basic architecture for knowledge sharing through a special class of agents called communication facilitators which coordinate the interactions of other agents.



state	For input formula F reasoning system R has
`proof`	found a proof for F
`model`	constructed a model for F
`mcm`	constructed models for both F and $\neg F$
`counter-model`	constructed a model for $\neg F$
`counter-proof`	found a proof for $\neg F$.
`valid`	determined that F is valid by some method
`properly-sat`	determined that F is neither valid nor unsatisfiable
`unsatisfiable`	determined that F is unsatisfiable ($:\Longleftrightarrow \neg F$ valid)
`satisfiable`	determined that F is satisfiable (has a model)
`negation-satisfiable`	determined $\neg F$ is satisfiable (has a model)
`determined`	determined one of the above

Figure 3. States of Mechanized Reasoning Systems

2.1 Reasoning Services in MathWeb

A MathWeb reasoning service is a mechanized reasoning system that tries to determine whether a given logical formula is valid (satisfied by all models), satisfiable (satisfied by some models), or unsatisfiable (not satisfied by any models). Automated theorem provers typically try to determine validity (theorem-hood) of a formula F[4] by finding a proof for F, or trying to refute the satisfiability of its negation $\neg F$. Model generators try to follow a dual approach, they try to construct a model for a formula F; if they succeed, then F is shown to be satisfiable, if they fail, exhausting all possibilities, then F is unsatisfiable.

[4]This will in general be of the form $A_1 \wedge \ldots \wedge A_n \Rightarrow C$, where the A_i are the assumptions (e.g. supplied by some background theory) and C the conclusion.

```
<achieve sender="A"
         receiver="kqml-xml://mathweb.org#atp"
         reply-with="id1"
         language="OpenMath">
   <OMOBJ><OMA><OMS cd="reasys" name="determined">F</OMA></OMOBJ>
</achieve>

<subscribe sender="A"
           receiver="kqml-xml://mathweb.org#atp"
           reply-with="id2"
           language="OpenMath">
   <ask-if sender="A"
           receiver="kqml-xml://mathweb.org#atp"
           reply-with="id3"
           language="OpenMath">
     <OMOBJ><OMA><OMS cd="reasys" name="valid">F</OMA></OMOBJ>
     <OMOBJ><OMA><OMS cd="reasys" name="proper-sat">F</OMA></OMOBJ>
     <OMOBJ><OMA><OMS cd="reasys" name="unsatisfiable">F</OMA></OMOBJ>
   </ask-if>
</achieve>
```

Figure 4. Querying a reasoning System for semantic status of F

In MathWeb we use the simple general hierarchy of states of mechanized reasoning systems to interact with the systems in KQML, which is shown in Figure 3. With this ontology, it is simple to communicate with all sorts of automated reasoning systems in KQML. For instance, the messages in Figure 4 are the normal way to request a judgment about the theorem-hood of a formula F.

Upon receiving the first message, the service `atp` at `mathweb.org`,[5] will try to determine the semantic status of the formula F by, e.g., concurrently starting one or several theorem provers and model generators in order to achieve the determined status. Upon receiving the `subscribe` message, it responds with an appropriate message (as if processing the `ask-if` message, most likely with a `tell` message) immediately after the status has been determined.

Querying an automated theorem proving system for a proof is similar, only employing an `ask-one` message using the more concrete statuses `model`, `proof`, `counter-model`, `counter-proof`, or `mcm`. These are defined in a special OpenMath content dictionary `reasys` (CD, see the references in the OpenMath symbols `OMS` in Figure 4), which is available from `http://www.mathweb.org/omdoc/cd/reasys.ocd`.

[5] It speaks the XML representation of KQML described in this paper, as we see by the prefix `xml-kqml`, so the representation of the messages is appropriate.

3 An Overview of OMRS

An OMRS specification consists of three layers: the *logic* layer (specifying the assertions manipulated by the system and the elementary deductions upon them), the *control* layer (specifying the inference strategies), and the *interaction* layer (specifying the interaction of the system with the environment). Notice that this layering allows for an additional and complementary way to structure the specifications w.r.t. the standard approach based on modularity. This domain-specific feature of the OMRS specification framework is fundamental to cope with the complexity of functionalities provided by state-of-the-art implementations.

The Logic Layer. The logic layer of an OMRS specification describes the assertions manipulated by the system and the elementary deduction steps the system performs upon such assertions. For example, a resolution-based theorem prover may manipulate first-order clauses by resolving and factorizing them. As another example, a linear arithmetic decider may manipulate polynomial inequalities by cross-multiplications and sums. At the logical level, the computations carried out by the system amount to constructing and manipulating structures consisting of assertions connected through elementary deduction steps (like proof trees). The key concept of the logic layer is that of reasoning theory. Roughly speaking, a *reasoning theory* (*RTh*) [16] consists of a set of *sequents* (i.e., assertions) and a set of *inference rules* over such sequents. An RTh defines a set of *reasoning structures*, i.e. graphs labeled by sequents and rules. The notion of reasoning structure generalizes the standard concept of derivation so as to capture, e.g., provisional reasoning and sub-proof sharing.

The Control Layer. Most real-world systems carry out their control strategies by making use of non-logical information, used exactly for control purposes. Examples of such control information are some history about how an assertion was produced, the number of times a certain inference step has been applied, the order in which some assertions must be selected for applying some reasoning steps, etc. Control information is used and modified during computation, at the same time as logical inferences are performed. The control layer of an OMRS specifies how reasoning systems manipulate the logic information, i.e., which strategies are used to select and apply the inference steps at each point of the computation. The concept of *Annotated Reasoning Theories* formalizes the control layer: It accounts for the simultaneous manipulation of logic and control information. More precisely, an annotated reasoning theory consists of a reasoning

theory and an erasing mapping. The sequents of the reasoning theory associated to the annotated reasoning theory are (annotated) sequents over an extended syntax which encode both logic and control information; the inference rules specify how such information is manipulated by the reasoning system. Finally, the erasing mapping specifies what is the logical content of the annotated sequents. The interested reader is urged to see [1] for a complete discussion of the control layer of the OMRS framework.

Both reasoning theories and annotated reasoning theories can be glued together yielding composite reasoning theories and annotated reasoning theories respectively.

The Interaction Layer. The OMRS interaction layer specifies the interaction of the reasoning system with the environment. At present only exploratory work has been carried out on this (see, e.g., [29, 3]) and a rigorous development of the interaction layer is part of the future work. The work described in this paper is a first exploratory step towards the definition of the interaction level in OMRS.

4 Constraint Contextual Rewriting as a Case Study

We show the applicability of our ideas by specifying a distributed version of Constraint Contextual Rewriting (CCR, for short) [2, 4]. We do this by first providing an OMRS specification of CCR (adapted from [1]) and then by describing an agent-oriented architecture for CCR obtained by turning the decision procedure into an autonomous agent which interacts with the simplifier via KQML performatives. We will show that the OMRS specification plays a fundamental role in the specification of the agent-oriented architecture.

CCR extends traditional conditional rewriting by exploiting the functionalities of a decision procedure as described in [4, 2]. We illustrate CCR by means of a simple example. Let us consider the problem of simplifying the clause $a < 1 \vee f(a) = a$ using the following fact as a conditional rewrite rule:

$$x > 0 \Rightarrow f(x) = x \tag{1}$$

where $<$ and $>$ are the the standard 'less-than' and 'greater-than' relations over the integers (resp.) and f is an uninterpreted function symbol. The basic step of the simplification process is to select a literal (called the *focus literal*) from the clause and start rewriting it, while assuming the negation of the remaining literals (called the *context*). For the clause above, let $f(a) = a$ be the focus literal and $\{a \not< 1\}$ be the context. Application of

(1) turns the focus literal into the identity $a = a$, under the proviso that the instantiated condition, namely $a > 0$, is entailed by the context. This can be established by means of a decision procedure for linear arithmetics by asking the decision procedure to check the satisfiability of the sets of literals obtained by adding the negation of the condition to the context, namely $\{a \not> 0, a \not< 1\}$. The simplification activity concludes that the identity $a = a$ is *true* by rewriting.

4.1 An OMRS Specification of CCR

An OMRS specification of CCR can be given by an annotated reasoning theory resulting from the combination of three annotated reasoning theories[6] each modeling a distinct reasoning module: the top level simplification loop (simp), the rewrite engine (cr), and the decision procedure (cs). The functional dependencies between the modules are depicted in Figure 5.(a). simp takes a clause (cl) and returns a simplified clause (cl'). cr performs conditional rewriting on the input literal (l) by using cs as rewriting context and returns a rewritten literal (l'). cs takes a conjunction of literals cnj and a context cs as input and returns a new context cs' obtained by extending cs with the literals in cnj.

The annotated reasoning theory for the overall simplification activity is the following (for details, see [1]). The sequents have the following forms:

- $cl \rightarrow_{\mathsf{simp}} cl'$ asserts that clause cl' (modeled as a finite set of literals) is the result of simplifying clause cl,
- $cs :: l \rightarrow_{\mathsf{cr}} l'$ asserts that literal l' is the result of rewriting l using cs (also called *constraint store*) as context,
- $cnj :: cs \rightarrow_{\mathsf{cs}} cs'$, asserts that cs' is the result of extending cs with the literals in cnj, cs-init(cs), asserts that cs is the "empty" constraint store, cs-unsat(cs) asserts that cs is an inconsistent (w.r.t. the theory decided by the decision procedure) constraint store, and
- $l \prec l'$ asserts that the literal l is smaller than the literal l' w.r.t. a simplification ordering (see, e.g., [11] for a definition).

Let R be a set of conditional equations of the form $cnj \Rightarrow (s = t)$, where cnj is a set of literals intended conjunctively. The rules of the annotated

[6] For the lack of space we confine ourselves to specifying the control layer and provide an informal explanation of the logic content. The interested reader may consult [1] for the details.

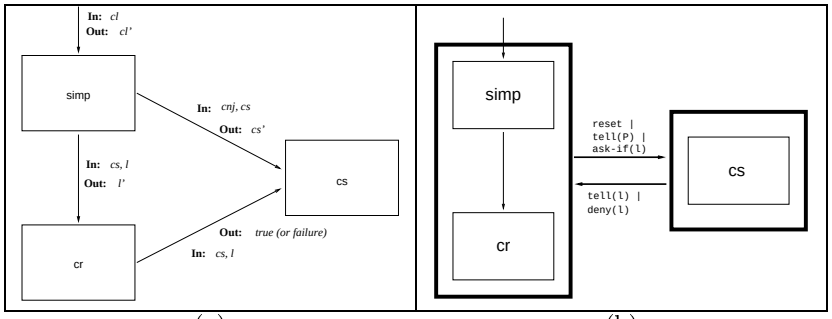


Figure 5. A control-level specification (a) and an agent-based architecture (b) of the case study

reasoning theory are the following:[7]

$$\frac{}{cl \cup \{true\} \rightarrow_{\mathsf{simp}} \{true\}}\ \texttt{cl-true} \qquad \frac{}{cl \cup \{false\} \rightarrow_{\mathsf{simp}} cl}\ \texttt{cl-false}$$

$$\frac{\mathsf{cs\text{-}init}(cs_0) \quad \overline{cl} :: cs_0 \rightarrow_{\mathsf{cs}} cs \quad cs :: l \rightarrow_{\mathsf{cr}} l'}{cl \cup \{l\} \rightarrow_{\mathsf{simp}} cl \cup \{l'\}}\ \texttt{cl-simp}$$

Rules `cl-true` and `cl-false` specify how to simplify a clause when $true$ and $false$ are in it, respectively. Rule `cl-simp` says that a literal l in a clause $cl \cup \{l\}$ can be replaced by a new literal l' obtained by rewriting l in context cs (premise $cs :: l \rightarrow_{\mathsf{cr}} l'$), where cs is obtained by extending the empty rewriting context cs_0 (premise $\mathsf{cs\text{-}init}(cs_0)$) with the negated literals in cl (premise $\overline{cl} :: cs_0 \rightarrow_{\mathsf{cs}} cs$).

$$\frac{\{\overline{l}\} :: cs \rightarrow_{\mathsf{cs}} cs' \quad \mathsf{cs\text{-}unsat}(cs')}{cs :: l \rightarrow_{\mathsf{cr}} true}\ \texttt{cxt-ent}$$

$$\frac{cs :: cnj\sigma \rightarrow_{\mathsf{cr}} \emptyset \quad l[t\sigma]_u \prec l[s\sigma]_u}{cs :: l[s\sigma]_u \rightarrow_{\mathsf{cr}} l[t\sigma]_u}\ \texttt{crew}$$

for each conditional rewrite rule $cnj \Rightarrow (s = t)$ in R. $cs :: cnj \rightarrow_{\mathsf{cr}} \emptyset$ abbreviates $cs :: l \rightarrow_{\mathsf{cr}} true$, for all $l \in cnj$ and $s[l\sigma]_u$ denotes the expression obtained from s by replacing the sub-expression at position u with $l\sigma$. Rule `cxt-ent` asserts that a literal l can be rewritten to $true$ in the rewrite context cs if the result of extending cs with the negation of l yields

[7] If l is an atomic formula, then $\overline{l}$ stands for $\neg l$; if l is a negated atom of the form $\neg m$, then $\overline{l}$ abbreviates m. If cl is a set of literals, then $\overline{cl}$ abbreviates $\{\overline{l} \mid l \in cl\}$.

`reset`(A,B)
Pre:
Post: $bel(B, c)$ iff $c \in cs_0$ where cs_0 is s.t. cs-init(cs_0)

Figure 6. Semantics of the `reset` performative

an inconsistent rewrite context (premise cs-unsat(cs')). Finally, rule `crew` says that the sub-expression $s\sigma$ at position u in the expression l can be rewritten to $t\sigma$ in the rewriting context cs if $cnj \Rightarrow (s = t) \in R$, σ is a ground substitution s.t. the literal $l[t\sigma]_u$ is $\prec$-smaller than $l[s\sigma]_u$ (premise $l[t\sigma]_u \prec l[s\sigma]_u$), and the instantiated conditions $cnj\sigma$ are entailed by the rewriting context cs (premise $cs :: cnj\sigma \rightarrow_{\mathsf{cr}} \emptyset$).

4.2 An Agent-Oriented Architecture for CCR

As depicted in the schema of Figure 5.(b), the agent-oriented architecture for CCR consists of two agents which interact by exchanging KQML messages. The agent on the left of Figure 5.(b) encapsulates the top level simplification loop (simp) and the rewriter (cr) whereas the agent on the right encapsulates the decision procedure (cs).[8]

The protocol consists of the repeated application of the following pattern of interaction. Whenever simp tries to apply rule `cl-simp` it initiates the interaction with cs by issuing on the channel a message containing the `reset` performative. (The semantics of the `reset` performative is given in Figure 6.) This has the effect of initializing the constraint store of cs (cf. precondition cs-init(cs_0) in `cl-simp`). Next, simp sends cs the set of literals occurring in the context via the message `tell`($\overline{cl}$)[9] (cf. precondition $\overline{cl} :: cs_0 \rightarrow_{\mathsf{cs}} cs$ in `cl-simp`) and finally it asks cr to rewrite the focus literal (cf. precondition $cs :: l \rightarrow_{\mathsf{cr}} l'$). cr rewrites the input literal using the available rewrite rules and in doing this it may ask cs to determine whether the current focus literal is entailed by the context via the message `ask`(l). In reply to this request cs sends back a message of the form `tell`(l) or of the form `deny`(l). (Notice that cr may query cs also when it is trying to establish the conditions of a conditional rewrite rule.)

In order to complete the description we must specify how the *bel* predicate is interpreted by the agents. Firstly we require that the decision

[8] For simplicity, we consider only two agents since we are interested in the interplay between the 'simplification' activity (namely clause simplification and rewriting) with the logical services provided by the decision procedure. However, the proposed methodology can easily be adapted to specify agent architectures with three or more agents.

[9] We omit the first two arguments of the performatives (namely the 'sender' and the 'recipient') whenever their identity can be inferred from the context.

procedure trusts the simplifier. This is formalized by the following axiom:

$$\forall c.(bel(\mathsf{simp}, c) \Rightarrow bel(\mathsf{cs}, c))$$

This fact allows the decision procedure to extend its own knowledge base using the information issued by the simplifier via the `tell` performatives. Secondly we must specify the inference capability of the decision procedure. This is done by means of the following axiom:

$$\forall p.\forall cs.\forall cs'.((bel(\mathsf{cs}, p) \wedge bel(\mathsf{cs}, cs) \wedge bel(\mathsf{cs}, p :: cs \rightarrow_{\mathsf{cs}} cs')) \Rightarrow bel(\mathsf{cs}, cs'))$$

where $bel(\mathsf{cs}, cs)$ abbreviates $\bigwedge\{bel(\mathsf{cs}, c) : c \in cs\}$ and $bel(\mathsf{cs}, p :: cs \rightarrow_{\mathsf{cs}} cs'))$ states the provability of the sequent $p :: cs \rightarrow_{\mathsf{cs}} cs'$ in the annotated reasoning theory of Section 4.1.

5 Implementation and Interfaces

How can the ideas presented in this paper help with the implementation and management of MathWeb? A general interaction protocol based on Internet standards transforms closed architectures like the MathWeb-SB [14] and the Logic Broker Architecture [5] systems into an open MathWeb architecture. Currently, the former uses the distributed programming features provided by the mOZart programming language [27] for communication, while the latter takes advantage of the communication functionality provided by Corba [9]. As a consequence, mathematical services either have to be embedded into a mOZart agent wrapper, or have to implement a Corba interface, or have to do both, if they want to communicate with services that are not present on both architectures.

In order to achieve a joint system that reuses much of the current functionality in a Kqml-based architecture, it is sufficient to augment each of the systems above by an agent that does the Kqml-communication and serves as a bridge. This agent is a Kqml facilitator agent that listens to a given network port (by default the OpenMath port 1473) and relays Kqml messages to the other agents in its architecture (by mOZart or Corba communication). This allows to reuse the existing implementations and internal communication among agents as well as providing a standardized interface to the Internet.

Note that this also allows other implementations than the ones listed above to participate in MathWeb as long as they implement the same outward appearance (e.g. Kqml, OMRS, and OMDoc). To simplify Kqml message passing, we will for the moment identify agent names with transport

addresses, since agent mobility seems not to be a problem in MathWeb. Agent names in MathWeb are quadruples of the form

$$\langle method \rangle : //\langle machine \rangle : \langle port \rangle \# \langle agent \rangle$$

that resemble URLs. For example the name of the broker agent would be `kqml-xml://mathweb.org:1473#broker`.

Since OMDoc and OpenMath use an Xml representation for mathematical objects, a first step for using Kqml in MathWeb is to supply an Xml encoding of Kqml. We have set up an Xml document type definition for Kqml (see `http://www.mathweb.org/omdoc/dtd/kqml.dtd`) based on the 1997 Kqml proposal by Finin and Labrou [25]. Finally, MathWeb-SB supports the commonly-used presentation-layer (see Figure 1) transport protocols `xml-rpc`, `http:get/put`, and sockets that can be used for `kqml-xml` message passing.

6 Conclusion

We have laid down the first ideas for implementing a communication protocol for reasoning services using Kqml and OMRS. The former provides the high-level performative- and message-layers, while the latter gives the specification infrastructure for determining (and for the agents to reason about) the meaning of interactions of reasoning services. Together with OMDoc [23] as a content language, this gives a suitable basis for communication of mathematical services in MathWeb [26].

The motivation for this paper and the general approach taken comes from our experience with the MathWeb-SB [14] and the Logic Broker Architecture [5] systems, and the perceived need for a standardized interaction layer. Both systems have a largely ad-hoc set of interaction primitives, following the needs of the growing systems. Conceptually it is clear, that all of these primitives can be mapped into Kqml, if we provide specification schemata for the internal states of reasoning systems, which we have started in this paper. The next step will be to implement the communication by `kqml-xml`. We have already implemented a `kqml-xml` interface for MathWeb. So it only remains to develop a `kqml-xml`-aware broker service and a Kqml-xml/Corba bridge.

For other MathWeb services, such as mathematical knowledge bases (e.g. the MBase system [21]) or symbolic computation systems, a corresponding ontology must still be developed, in order to access them via Kqml. For the former, the problem will be relatively simple as the Kqml views agents a virtual knowledge bases anyway, for the latter, a suitable variant of OMRS has been presented in [17].

References

[1] Alessandro Armando, Alessandro Coglio, Fausto Giunchiglia, and Silvio Ranise. The Control Layer in Open Mechanized Reasoning Systems: Annotations and Tactics. To appear in [20], 2000.

[2] Alessandro Armando and Silvio Ranise. Constraint Contextual Rewriting. In R. Caferra and G. Salzer, editors, *Proc. of the 2nd Intl. Workshop on First Order Theorem Proving (FTP'98)*, pages 65–75, 1998.

[3] Alessandro Armando and Silvio Ranise. From Integrated Reasoning Specialists to "Plug-and-Play" Reasoning Components. In Calmet and Plaza [7], pages 42–54.

[4] Alessandro Armando and Silvio Ranise. Termination of Constraint Contextual Rewriting. In H. Kirchner and Ch. Ringeissen, editors, *Proceedings of the 3rd International Workshop on Frontiers of Combining Systems, FroCoS'2000*, pages 47–61. Springer LNAI 1794, 2000.

[5] Alessandro Armando and Daniele Zini. Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture. In A. Poggi, editor, *to appear on the Proceedings of the AI*IA-TABOO Joint Workshop 'From Objects to Agents: Evolutionary Trends of Software Systems'*, Parma, Italy, May 29–30, 2000. (See also the article in this volume)

[6] P. G. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and Integration of Theorem Provers and Computer Algebra Systems. In Calmet and Plaza [7], pages 94–106.

[7] Jaques Calmet and Jan Plaza, editors. *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)*, Springer LNAI 1476, 1998.

[8] Olga Caprotti and Arjeh M. Cohen. Draft of the Open Math standard. The Open Math Society, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.

[9] The Object Management Group. The Common Object Request Broker Architecture. http://www.corba.org/.

[10] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham.

The Prosper Toolkit. In *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS-2000*, Springer LNCS, 2000.

[11] N. Dershowitz and J.P. Jouannaud. Rewriting systems. In *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier Publishers, Amsterdam, 1990.

[12] T. Finin and R. Fritzson. KQML — a Language and Protocol for Knowledge and Information Exchange. In *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*, pages 127–136, Seattle, WA, USA, 1994.

[13] Andreas Franke, Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, and Volker Sorge. Agent-Oriented Integration of Distributed Mathematical Services. *Journal of Universal Computer Science*, 5:156–187, 1999.

[14] Andreas Franke and Michael Kohlhase. System Description: MATHWEB, an Agent-Based Communication Layer for Distributed Automated Theorem Proving. In Harald Ganzinger, editor, *Proceedings of the 16th Conference on Automated Deduction*, pages 217–221. Springer LNAI 1632, 1999.

[15] D. Fuchs and J. Denzinger. Knowledge-Based Cooperation between Theorem Provers by Techs. Seki Report SR-97-11, Fachbereich Informatik, Universität Kaiserslautern, 1997.

[16] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories: Towards an Architecture for Open Mechanized Reasoning Systems. Technical Report 9409-15, IRST, Trento, Italy, 1994.

[17] K. Homann. *Symbolisches Lösen mathematischer Probleme durch Kooperation algorithmischer und logischer Systeme.* PhD thesis, Unversität Karlsruhe, 1996. DISKI 152, Infix; St. Augustin.

[18] N. R. Jennings and M. Wooldridge. *Handbook of Agent Technology*, chapter Agent-Oriented Software Engineering. AAAI/MIT Press, 2000.

[19] Nicholas R. Jennings and Michael J. Wooldridge, editors. *Agent Technology : Foundations, Applications, and Markets.* Springer, 1998.

[20] T. Jebelean and Alessandro Armando, eds. *J. Symbolic Computation*. Special Issue on Integrated Symbolic Computation and Automated Deduction, 2000.

[21] M. Kohlhase and A. Franke. Mbase: Representing Knowledge and Context for the Integration of Mathematical Software Systems. to appear in [20], 2000.

[22] Michael Kohlhase. OMDoc: Towards an Internet Standard for the Administration, Distribution and Teaching of Mathematical Knowledge. In *Proceedings of AI and Symbolic Computation, AISC-2000*, Springer LNAI, 2000. in press.

[23] Michael Kohlhase. OMDoc: Towards an OPENMATH Representation of Mathematical Documents. Seki Report SR-00-02, FB Informatik, Universität des Saarlandes, 2000. `http://www.mathweb.org/omdoc`.

[24] Y. Labrou and T. Finin. A Semantics Approach for KQML—A General Purpose Communication Language for Software Agents. In *Third International Conference on Information and Knowledge Management (CIKM'94)*, November 1994.

[25] Yannis Labrou and Tim Finin. A Proposal for a New KQML Specification. Tech Report TR CS-97-03, University of Maryland, Baltimore County, 1997.

[26] The MathWeb group. MathWeb.org: Supporting Mathematics on the Web. http://www.mozart-oz.org/.

[27] The Oz group. The MOZART Programming System. http://www.mozart-oz.org/.

[28] M. Dewar ed. Special Issue on OpenMath Bulletin of the ACM Special Interest Group on Symbolic and Algebraic Mathematic (SIGSAM), 2000, in press.

[29] C. Talcott. Reasoning Specialists Should Be Logical Services, Not Black Boxes. In *Workshop on Theory Reasoning in Automated Deduction (CADE12)*, pages 1–6, 1994.