Journal of Universal Computer Science, vol. 5, no. 3 (1999), 156-187
submitted: 1/11/98, accepted: 15/3/99, appeared: 28/3/99 © Springer Pub. Co.

Agent-Oriented Integration of Distributed Mathematical Services

Andreas Franke
(*FB Informatik, Universität des Saarlandes, Germany*
afranke@ags.uni-sb.de)

Stephan M. Hess
(*FB Informatik, Universität des Saarlandes, Germany*
http://www.ags.uni-sb.de/~hess
hess@ags.uni-sb.de)

Christoph G. Jung
(*MAS Group, DFKI GmbH, Germany*
http://www.dfki.de/~jung
jung@dfki.de)

Michael Kohlhase
(*FB Informatik, Universität des Saarlandes, Germany*
http://www.ags.uni-sb.de/~kohlhase
kohlhase@ags.uni-sb.de)

Volker Sorge
(*FB Informatik, Universität des Saarlandes, Germany*
http://www.ags.uni-sb.de/~sorge
sorge@ags.uni-sb.de)

Abstract: Real-world applications of automated theorem proving require modern software environments that enable modularisation, networked inter-operability, robustness, and scalability. These requirements are met by the *Agent-Oriented Programming* paradigm of *Distributed Artificial Intelligence*. We argue that a reasonable framework for automated theorem proving in the large regards typical *mathematical services* as autonomous agents that provide internal functionality to the outside and that, in turn, are able to access a variety of existing external services.

This article describes the MathWeb architecture that encapsulates a wide range of traditional mathematical systems each into a social agent-shell. A *communication language* based on the *Knowledge Query and Manipulation Language* (KQML) is proposed in order to allow conversations between these mathematical agents. The individual *speech acts* of their conversations are about performances of the encapsulated services. The *objects* referred by these speech acts are mathematical objects, formulae in various logics, and (partial) proofs in different calculi whose formalisation is done in an extension to the OpenMath standard. The result is a flexible framework for automated theorem proving which has already been implemented to a large extent in the context of the Ωmega proof development system.

1 Introduction

The work reported in this article originates in the effort to develop a practical mathematical assistant that integrates external deductive components. The ΩMEGA system [Benzmüller *et al.*, 1997] is an interactive, plan-based deduction system with the ultimate goal of supporting theorem proving in mainstream mathematics and mathematics education. To provide the necessary reasoning and symbolic computation facilities it incorporates the first-order theorem provers BLIKSEM, EQP, OTTER, PROTEIN, SPASS, WALDMEISTER (see [Sutcliffe and Suttner, 1997] for references), the two higher-order theorem provers TPS [Andrews *et al.*, 1996] and $\mathcal{LEO}$ [Benzmüller and Kohlhase, 1998], and the computer algebra systems MAPLE, MAGMA, GAP and μCAS (see [Kerber *et al.*, 1998] for references).

Traditional deduction systems, such as the ones integrated into ΩMEGA, as well as today's tactical theorem provers, such as ISABELLE [Paulson, 1994] or NQTHM [Boyer and Moore, 1979], are monolithic systems. They either work like compilers – reading a problem file and writing proof and log files after successful computation – or like programming environments featuring their own command interpreter or graphical user interface. Driven by the complexity of real-world reasoning problems and practical considerations in designing and interacting with the system, we have seen a rapid move towards integrative frameworks combining various external reasoners [Denzinger, 1993; Benzmüller *et al.*, 1997; Dahn, 1997] and computation systems [Clarke and Zhao, 1992; Harrison and Théry, 1993; Ballarin *et al.*, 1995; Kerber *et al.*, 1998].

Ideally, the reasoning modules in the ΩMEGA system interact with each other to complete open subgoals during the development of a proof. This can be initiated and supervised on-line by the user. This can be also guided by the ΩMEGA system itself, for instance during proof planning in order to expand a given proof plan to a full proof. Unfortunately, it is not always clear in advance, which prover is best suited for the problem at hand. Furthermore, the user could be asked to support the system with additional knowledge. Thus, ΩMEGA will call several 'services' in parallel in order to maximise the likelihood of success and minimise the time the user has to spend waiting for the system. The proprietary proofs found by these systems are transformed into the internal format of the ΩMEGA system; again, this transformation process should run in parallel to the ongoing user interaction.

The role of the mathematical assistant in particular, but also of general applications of theorem proving in the large, for instance in program verification [Hutter *et al.*, 1996], call for an open and distributed architecture. In such an architecture, the developer of a deduction system or a mathematical tool upgrades it to a so-called *mathematical service* [Homann and Calmet, 1996] by providing it with an interface to a common *mathematical software bus* [Calmet and Homann, 1997]. That is, it provides the mathematical service instead of the software itself. In the context of the ΩMEGA system, we have implemented and experimented with such a network design, where the integrated theorem provers and mathematical tools interact distributed over the Internet and can be dynamically added to and subtracted from the coordinated reasoning repertoire of the complete system. The possible benefits of such an approach to semi-automated proof development are:

Modularisation The more external reasoners a system like the ΩMEGA system integrates the heavier the burden of installing and maintaining them gets. For instance, the kernel of ΩMEGA alone is a rather large system (roughly 17 MB of COMMON LISP code for the main body in the current version), its successful installation depends on the presence of (proprietary) compilers or interpreters. This situation is similar for the other reasoning systems integrated into the ΩMEGA system, which come from numerous different original sources. For the user it is a burden to install and understand the complete system, for the developers it is a tedious task to port the system to commonly available compilers. Thus providing a mathematical service instead of software encapsulates related functionality into re-usable components and eases the maintenance of the particular modules and environments built upon them at the cost of requiring a constant pool of hardware resources.

Deduction systems are among the most complex existing AI programs, they are typically developed by more than one individual and the respective components require specialised know-how that is nowadays impossible to acquire for a single person. The equivalent is true for Computer Algebra systems that exist in a vast variety from multipurpose to very specialised ones. Both user and developer can hardly distinguish which system is best suited for a particular task, let alone being able to use all different systems. Thus a modular architecture of mathematical services allows the focused and independent development in specialised research groups, for specialised application areas, and with specialised techniques.

Inter-Operability Having a means of modularisation, the requirement appears of being able to easily put together a complete and working system out of heterogeneous components. Having a common platform of exchanging services across the network makes components inter-operable: they are able to provide additional functionality for the system as a whole and, in turn, are provided with additional services in order to perform their service far more efficiently. For inter-service exchange of data, it is important to even take possible, but yet not existing components into account, i.e., the interaction scheme should be generic and *open*. This accelerates the availability of new developments, because it avoids ubiquitous re-engineering.

Robustness Fixed software architectures pose the problem of failure handling, e.g., a typical proof system with a static topology will not work if one of its integral mathematical modules does not function or has to undergo maintenance. A dynamic, decentralised network architecture provides the ability of bringing together available and partially redundant components on the fly. Temporarily shutting off a particular mathematical service for maintenance purposes thus should not do any harm.

Scalability Finally, the performance aspect of theorem proving in the large is addressed by a distributed architecture. In local computer networks, the situation is quite common that users have relatively low-speed machines on their desktop, whereas some high-speed servers operate freely accessible in the background. Running, e.g., the user interface on the local machine uses the local resources that are 'close to the relevant data' and sufficient for this task while the more powerful servers can be fully exploited for the really complex task of actually proving theorems. A flexible, dynamic topology is the key to optimally adapt to changing computational resources, thus increases the scalability of theorem proving.

Indeed, these desiderata comply remarkably well with the aims of the *Agent-Oriented Programming* paradigm developed in the field of *Distributed Artificial Intelligence*: Intelligent agents are self-interested, autonomous service programs which flexibly interact in a shared, also human-inhabited environment by means of *communication*. The agent metaphor has been successfully applied to a spectrum of sophisticated software problems ranging from 'hardbots' in robotics and telematics to 'softbots' in user assistance systems.

Consequently, the present article proposes this perspective as the basis of the MathWeb architecture which generalises the work done in ΩMEGA: MathWeb-agents 'incarnate' particular mathematical services and possess a (partial) representation of the service network. MathWeb-agents share a standardised communication language to talk about mathematical objects, formulae, and proofs (*objects* of communication) and to address the services which they provide (*speech acts* of communication). MathWeb agents are *reactive* in the sense that they are steadily interacting with users and other software agents working on shared proofs and mathematical computations. They are *pro-active* in that they adopt and autonomously work on particular mathematical goals. And they are *social* in the sense that they request other agents or even the human user to support the successful execution of their services.

Thus theorem proving in MathWeb is the joint effort of a society (a *multi-agent system*) of communicating mathematical agents. We propose MathWeb as a convenient design stance to enable modularisation, networked inter-operability, robustness and scalability in theorem proving. In particular MathWeb does **not** in itself aim at improving the **expressivity** of theorem proving per se, as other approaches to cooperating theorem provers do (see e.g. [Denzinger and Dahn, 1998] and the references therein). This may be an ultimate effect of providing the distribution layer in MathWeb, but the current paper does not make any concrete claim in this direction.

1.1 Structure of the Article

We start with a motivational example showing how it is processed by the hybrid ΩMEGA system in Section 2. From these considerations, the requirement of finding a suitable methodology for distributing mathematical services immediately arises. This software methodology is given by the agent metaphor of *Distributed Artificial Intelligence* and corresponding efforts for building domain-independent communication languages, such as the *Knowledge Query and Manipulation Language* (KQML) (Section 3). The MathWeb architecture for automated theorem proving (Section 4) thus extends ΩMEGA into an open and distributed society of mathematical agents which use KQML *performatives* (speech act types) to address their services. MathWeb agents are equipped with a standardised *content* language OpenProof (Section 5) derived from the OpenMath specification to talk about mathematical objects, formulae, and proofs. At hand of a collection of existing (and planned) mathematical services, we demonstrate that MathWeb is a powerful agent-oriented tool for their integration.

1.2 Related Work

In [Fisher and Ireland, 1998], Fisher and Ireland propose an agent-based approach to proof planning that is motivated by a fine-grained parallelisation of the proof planning process more than the distribution aspect. They propose a society of agents that are organised by a contract net architecture, building on earlier studies of Fisher [Fisher, 1997] on agent-based theorem proving.

Calmet and Homann present a framework for establishing the semantics of intimately integrated deduction and computation systems [Homann, 1996; Homann and Calmet, 1996]. In a servicing architecture like the one described in this paper, the semantics of the protocol employed in the communication is not a correctness problem, since our approach assumes that proofs are communicated, so that the initiator of a reasoning task can always collect the partial proofs and verify the correctness of the final resulting proof if he does not trust the mathematical services.

To our knowledge, only three distributed theorem proving systems besides ΩMEGA have actually been implemented up to now. The modal-logic theorem prover from [Pitt, 1996] uses a trader model like the one realized in ΩMEGA. The ILF system [Dahn, 1997] connects to MATHEMATICA and some automated theorem provers in a simple master-slave model. A group of experimental systems centreing around the DISCOUNT theorem prover has been presented by [Denzinger *et al.*, 1997; D. Fuchs, 1997]. Their experiments explore a tight cooperation between the theorem provers that renders them as a group significantly more successful than any of them could be alone. The underlying TEAMWORK and TECHS approach to distribution (see [Denzinger and Dahn, 1998]) is probably the work closest to MATHWEB, but the emphasis was laid on supporting the particular cooperation model and not so much on standardisation and generality. In particular, MATHWEB would provide a drop-in replacement for their implementation.

By introducing a service-independent communication language based both on KQML [Finin and Fritzson, 1994] and OPENMATH [Abbot *et al.*, 1996], our approach is unique so far with respect to the consequent application of Shoham's *Agent-Oriented Programming* paradigm [Shoham, 1990] to Automated Theorem Proving. As such, it is the logical progression of our work on distributing the ΩMEGA system [Hess *et al.*, 1998; Siekmann *et al.*, 1998] and opens up the possibility for developing particular negotiation protocols. In general multi-agent system design, a similar stance has been taken by the MECCA architecture [Steiner, 1992].

2 Distributing Mathematical Services

In this section we introduce a small example to elaborate the principle of the hybrid ΩMEGA architecture [Benzmüller *et al.*, 1997] in order to motivate a software methodology for distributed mathematical services. We use a simple problem from Algebra — more precisely group theory — that states the equivalence of two different axiomatisations of a group. Both are rather common and can be found in most textbooks of group theory (cf. [Hall, 1959]):

Definition 2.1 *Let G be a non-empty set, then G together with binary operation '$\cdot$' is a group if the following properties hold:*

$\mathcal{G}$1) *For all $a, b \in G$ there is a $c \in G$ with $a \cdot b = c$.*
$\mathcal{G}$2) *For all $a, b, c \in G$ holds $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*

$\mathcal{G}$**3)** *There exists an $e \in G$ such that $e \cdot a = a$ and $a \cdot e = a$ for all $a \in G$.*
$\mathcal{G}$**4)** *For all $a \in G$ exists $x \in G$ such that $a \cdot x = e$ and $x \cdot a = e$.*

Definition 2.2 *Let G be a non-empty set, then G together with binary operation '$\star$' is a group if the following properties hold:*

$\mathcal{H}$**1)** *For all $a, b \in G$ there is a $c \in G$ with $a \star b = c$.*
$\mathcal{H}$**2)** *For all $a, b, c \in G$ holds $(a \star b) \star c = a \star (b \star c)$.*
$\mathcal{H}$**3)** *For all $a, b \in G$ exist uniquely determined $x, y \in G$ such that $a \star x = b$ and $y \star a = b$.*

To prove the equivalence of both definitions we have to infer the axioms of the second definition assuming that the first definition holds and vice versa. However, in both cases we have to decide in advance how we express the operation on one group definition by a suitable term involving the operation given in the other definition. This is generally a non-trivial task, however in the case of our example we can simply identify both operations with each other. The actual verification of the single axioms is then done by finding suitable combinations of the given equations.

For instance, we verify the first part of the existence of divisors in definition 2.2 by showing the existence and uniqueness of the solutions of equation $ax = b$ using definition 2.1: The existence of a suitable x is obvious by setting $x = a^{-1}b$, where a^{-1} denotes the inverse element introduced by $\mathcal{G}4$, and verifying that $ax = a(a^{-1}b) = (aa^{-1})b = eb = b$ holds. To show uniqueness we assume now that there exist two solutions $x, x' \in G$ of our original equation, then we have with $b = ax = ax'$ and multiplication with a^{-1} the uniqueness of solutions by: $a^{-1}b = x = x'$.

2.1 Formal Proof Development in Hybrid ΩMEGA

The equivalence of different axiomatisations of the same mathematical entity is a general problem that arises in the hybrid ΩMEGA system (Figure 1 shows only the components and the information flow which are relevant for our example) when the same entity is tried to be defined alternatively in ΩMEGA's knowledge base. A similar situation appears when ΩMEGA receives definitions from two separate knowledge bases, as depicted in Figure 1. The central component of ΩMEGA is the controller. It supervises the process of proving a theorem by handling requests to knowledge bases, distributing subproblems to reasoning components and accepting user input via some user interface. To illustrate the processing of our equivalence problem, we assume that both group axiomatisations in the knowledge bases are given as higher order formulas in a typed Church λ-calculus [Andrews, 1986] with base types $\{o, \iota\}$:

$$\begin{aligned}\texttt{group-1} := \lambda G_{\iota\to o}.\lambda Op_{(\iota,\iota)\to\iota}.\ & \texttt{not-empty}(G) \wedge \texttt{closed-under}(G, Op) \wedge \\ & \texttt{associative}(G, Op) \wedge \exists e. G(e) \wedge \\ & \texttt{unit}(G, Op, e) \wedge \texttt{inverse-exists}(G, Op, e) \end{aligned} \quad (1)$$

$$\begin{aligned}\texttt{group-2} := \lambda G_{\iota\to o}.\lambda Op_{(\iota,\iota)\to\iota}.\ & \texttt{not-empty}(G) \wedge \texttt{closed-under}(G, Op) \wedge \\ & \texttt{associative}(G, Op) \wedge \\ & \texttt{divisors-exist}(G, Op). \end{aligned} \quad (2)$$

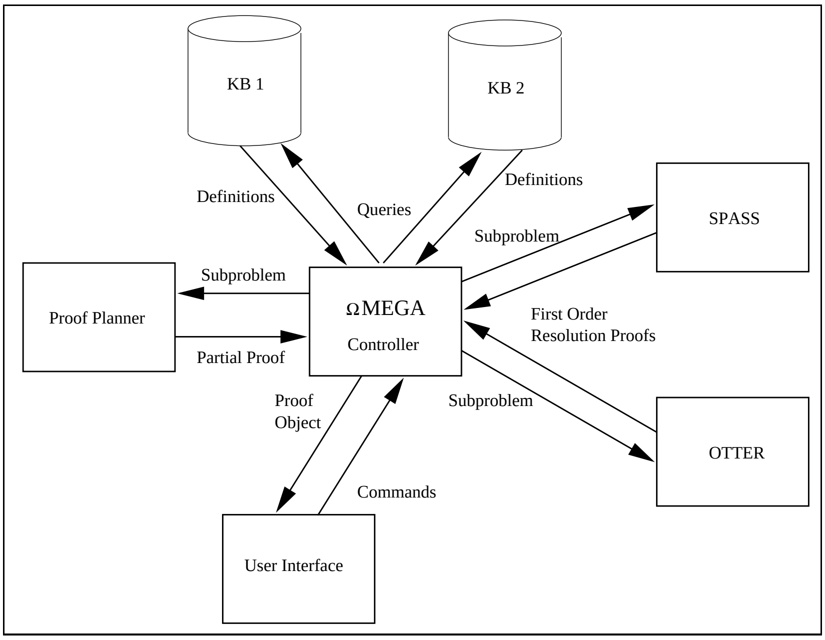

Figure 1: Distributed Mathematical Services in ΩMEGA

Both terms consist of conjunctions of high level concepts that are in turn defined in the respective knowledge bases using other λ-terms. All high level concepts directly correspond to a single axiom given in the informal definitions (this is of course indicated by the choice of names). Here we assume for simplicity that `not-empty`, `closed-under`, and `associative` represent the same concepts and are equally named in both knowledge bases. In order to compare both concepts, ΩMEGA tries to prove the equivalence of the given formalisations. This goal is specified by user interaction via the interface. The corresponding theorem is of the form:

$$\forall G_{\bullet}(\exists Op1_{\bullet}\texttt{group-1}(G, Op1) \equiv \exists Op2_{\bullet}\texttt{group-2}(G, Op2)) \tag{3}$$

ΩMEGA sends this theorem together with the retrieved λ-terms (1) and (2) to the proof planner. The planner uses a set of domain independent planning-operators (called *methods*) that it can employ to simplify the theorem. This set contains a particular method (cf. [Cheikhrouhou and Sorge, 1998]) that applies to formulae like the one given in (3). It splits the equivalence, expands the group definitions and partitions the proof into single subgoals. Each one of these subgoals contains one of the conjuncts given in (1) and (2). This method also introduces meta-variables for the existentially quantified variables, i.e., the two different operations defined on the group and the identity element.

There are other methods which are able to close some of the trivial subgoals, i.e., subgoals that directly correspond to formulas given as hypotheses.

For example, the properties `not-empty`, `closed-under`, and `associative` can be directly inferred for both axiomatisations. Thereby the methods compute possible instantiations for the introduced meta-variables. The planner finally returns a proof plan, containing the partial proof derived so far, together with the proposed instantiations for the meta-variables. In case of our example the planner would propose to instantiate both meta-variables for the two operations of the group with the same constant and the identity element of definition 1 with any arbitrary constant. Expanding the planning steps in ΩMEGA results in four remaining subgoals.

$$\mathcal{H} \vdash g(\overline{E}) \tag{4}$$

$$\mathcal{H} \vdash \mathtt{unit}(g, \overline{Op1}, \overline{E}) \tag{5}$$

$$\mathcal{H} \vdash \mathtt{inverse\text{-}exists}(g, \overline{Op1}, \overline{E}) \tag{6}$$

$$\mathcal{G} \vdash \mathtt{divisors\text{-}exist}(g, \overline{Op2}) \tag{7}$$

Here g denotes a constant instantiated for the universally quantified variable in (3) and the over-lined letters indicate the meta-variables introduced by the planner. Furthermore, $\mathcal{H}$ and $\mathcal{G}$ specify sets of hypotheses which correspond to the axiomatisations of definition 2.1 and 2.2 respectively.

In order to further treat the subgoals (4) through (7) ΩMEGA expands the high level concepts given both in the goal and the hypotheses, by fetching the appropriate definitions from the knowledge bases. These definitions are again λ-terms that can easily be substituted in the formulas which are subsequently β-normalised. For example the existence of inverses in a group corresponds to

$$\lambda G_{\iota\to o} . \lambda OP_{(\iota,\iota)\to\iota} . \lambda E_{\iota} . \exists F_{\iota\to\iota} . \forall X_{\iota} . G(X) \to OP(X, F(X)) = E \tag{8}$$

With all definitions expanded it is now possible to hand the remaining problems over to an automated theorem prover. In our example it suffices to give a single subproblem together with its expanded hypotheses lines to some automated theorem prover, such as OTTER [McCune and Wos, 1997] or SPASS [Weidenbach, 1997]. For this, ΩMEGA translates higher order syntax into first order and substitutes the meta-variables with the instantiations proposed by the planner. If the planner has proposed more than one possible instantiations of the meta-variables, the process of calling ATPs is iterated for all the instantiations until some proof could be found. If we have dependencies between subproblems, i.e. meta-variables need to be substituted with the same term in different subproblems, ΩMEGA keeps track of these meta-variables and compares instantiations given by the respective automatically generated subproofs. If different instantiations are returned, ΩMEGA tries to match or unify these, and if this fails ΩMEGA successively uses generated meta-variable substitutions of one subproblem on the dependent ones and tries to prove those subproblems again by calls to automated components. Eventually, if all this fails, the proof is left to the user. In our example, however, the proofs returned from the theorem provers are simply translated back into ΩMEGA's syntax and calculus and complete the proof.

During the whole process of proof construction a user can always monitor the progression of the proof and, if necessary, interfere and influence the next step. In our example, the expansion of definitions and the actual activation of theorem provers has to be confirmed by an ΩMEGA user.

2.2 Desiderata of a Methodology for Distributing Mathematical Services

The above example illustrates how work on a single problem can be shared between different components of a hybrid system. The system is basically built around the ΩMEGA controller as the central unit that cannot only deploy other systems but has deductive capabilities by itself, i.e. it can expand planning methods and definitions, compare meta-variable instantiations by means of unification, or apply single deduction steps indicated by the user. The advantages of such a system are that problems can be tackled that are beyond the reach of a single automated theorem prover. As displayed in Figure 1, however, the protocols that ΩMEGA mediates are proprietary ones and the architecture itself is static in a sense that single components, e.g., the controller, cannot be easily exchanged. For instance, it is a expendable task to simply substitute one first-order theorem prover for another since usually the syntax translator of the interface has to be redesigned. The same problem arises in other approaches concerned with the integration of two or several theorems provers (cf. [Felty and Howe, 1997; Slind *et al.*, 1998; Benzmüller and Sorge, 1998]) or theorem provers with decision procedures or Computer Algebra systems (cf. [Clarke and Zhao, 1992; Harrison and Théry, 1993; Ballarin *et al.*, 1995; Kerber *et al.*, 1998]) that generally do not follow a common paradigm, i.e., a similar input-output specification. Their solutions do heavily depend on the integrated systems.

The question how different theorem provers can be easily combined in a single environment that is flexible enough to handle both replacement and addition of systems has led to the concept of **O**pen **M**echanized **R**easoning **S**ystems [Giunchiglia *et al.*, 1996]. Within an OMRS, theorem provers can be viewed as easily replaceable plug and play components. The concept of OMRS has been generalised to **O**pen **M**athematical **E**nvironments [Homann and Calmet, 1995] where all kinds of *mathematical services* [Homann and Calmet, 1996] can be combined. It turned out that in order to handle a mathematical service (either a theorem prover or a Computer Algebra system) as a plug and play component the systems have to be at least separated into distinct components for control and logic or computation. Thus, it is practically impossible to integrate any monolithic system without redesigning major parts. Moreover, commercial systems where the sources are not available cannot be re-engineered and are therefore lost for an integration.

This inspired the extension of the latter architecture to cope with heterogeneous mathematical services (such as theorem provers, Computer Algebra systems, editors, display components, etc.). On the *mathematical software bus* [Calmet and Homann, 1997], connected services can exchange information by directly sending standardised mathematical objects to a specified service. Yet, the approach still has two major drawbacks: Firstly, all connected systems have to communicate in some standardised language. Although there have been some efforts to establish some standard for exchange of mathematical object lately (cf. [Caprotti, 1998; Ion, 1998]) these languages are still far from being general enough for a variety of possible services. A second drawback of the architecture is the principle of a software bus itself. Connected services need to know of other services or at least of a central directory (request broker) available on the software bus in order to send directed messages. To maintain this knowledge within each service or within a central directory is a difficult task for a freely expanding software bus that is for example distributed over the Internet. Furthermore, this architecture lacks robustness in a sense that if a connected service, especially the central directory, fails there are no means for the requesting service to redirect its query.

These considerations reveal a methodological challenge: which is the right software engineering metaphor to integrate a variety of mathematical services as particular modules? How is it possible to make these modules inter-operable, preferably over a global network, at the same time staying open for future enhancements? How do we support a dynamic architecture which is robust to the exchange or maintenance of embedded services and which is scalable to efficiently adapt to changing computational resources? As can be seen from our preceding critique, the listed desiderata are not fully addressed by *Distributed Object-Oriented Programming* paradigms, such as the *Common Object Request Broker Architecture* (CORBA) approach [Siegel, 1996].

3 Agent-Oriented Programming

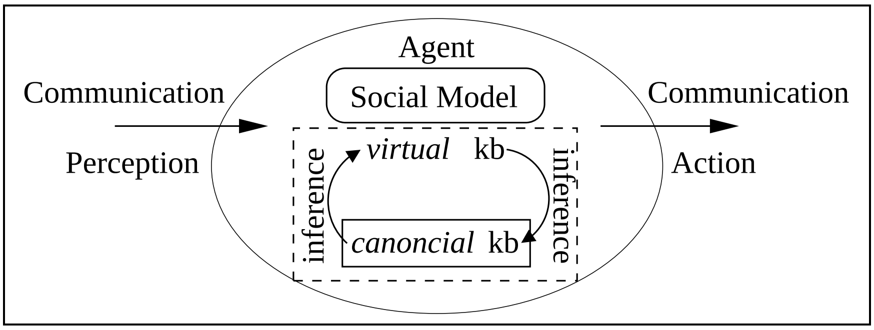

Figure 2: A (Social) Agent Architecture

Following Russell & Norvig [Russell and Norvig, 1995], the term agent describes a self-contained computational structure, i.e., a state and a corresponding calculation (Figure 2). This structure is encapsulated by a separate *environment* which the agent perceives through *sensors* and upon which the agent acts through *effectors*. The definition is close to the one of a *robot* which it generalises to software environments (*softbots*). Both physical and virtual environments share the requirements of local, decentralised control (modularisation), the handling of inherent complexity (scalability and robustness), and heterogeneous, open structures (inter-operability). The agent paradigm put forward by research in *Distributed Artificial Intelligence* is a novel combination of fundamental technologies from *Distributed Systems*, *Embedded Systems*, *Object-oriented Programming*, and *Artificial Intelligence* and seems to be the natural metaphor to manage these requirements.

3.1 Properties of Agents

Despite of the sometimes mentalistic terminology of DAI, agent properties are deeply rooted in purely technical concepts. Partly due to the broadened perspective and partly due to new insights into the agent as a *situated* entity, an enlarged set of key properties of agency proposed by Wooldridge & Jennings [Wooldridge and Jennings, 1995] is nowadays commonly agreed on (Other agent features which are researched are *mobility* and *veracity*.):

Autonomy: Agents are encapsulated, i.e., they should be able to perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and they should have a degree of control over their own actions and their own internal state. Autonomy is the focus of the agent definition given by Russell & Norvig [Russell and Norvig, 1995].

Responsiveness, Reactivity: Agents should respond in a timely fashion to changes which occur in their environment, i.e., they are *reactive*. Note that this does not necessarily entail real-time behaviour.

Pro-activeness and Deliberation: Agents should not simply act in response to their environment, but also exhibit *goal-directed* behaviour to take initiative where appropriate. We speak of *deliberative* abilities in this respect and presume *rationality* , i.e., from its current *belief*, the agent *decides* (chooses) *intentions* which are actions to achieve its goals. Furthermore, the agent avoids behaviour which he believes to conflict with them. Interestingly, regarding the agent's state as a knowledge base and its computation as a rational inference procedure (Figure 2) closely mirrors the image of a theorem prover. And in fact historically, the first agents were modelled as deductive/abductive inference systems.

Adaptivity: Agents should be able to modify their behaviour according to changing environmental and computational constraints to their functioning (*resources*, such as fuel, space, tools; time, memory). According to the more and more popular *bounded rationality* principle [Good, 1976; Simon, 1982], they should do that in an approximately optimal manner.

Social Ability: Agents should be able to interact, when they deem appropriate, with other artificial agents and humans in order to complete their own problem solving and help others with their activities. This requires that agents have, as a minimum, a means by which they can *communicate* their requirements to others and an internal, rational mechanism (*social model*) for deciding when social interactions are appropriate (Figure 2) — both in terms of generating appropriate requests and judging incoming requests. Social abilities are the key to design *open* systems in which heterogeneous information entities operate in a common framework upon different goals and on behalf of different users.

3.2 Agent Communication Languages: KQML

Shoham coined the term *Agent-Oriented Programming* [Shoham, 1990] as a software methodology in which softbots, such as the one depicted in Figure 2, are used to encapsulate arbitrary, traditional software applications, e.g., legacy systems. These agent-shells are able to interface and control the operation of the embedded services quite similarly to the way a knowledge base would operate.

On top, they introduce a social model referring to other service agents with which they comprise a society. The prominent means for the interaction between social agents in a functional service network turn out to be common *communication languages* which enable the agents to coordinate their behaviour, i.e., steer the embedded applications by exchanging beliefs, goals, and intentions. As a part of the fast-growing research threads in *Computer Science*, Shoham's work triggered a gamut of innovative software applications, e.g., in robotics, personal assistants, work-flow management, telecommunication, information retrieval, etc.

Artificial communication languages go back to philosophical and linguistic (especially pragmatics) observations into human language which they transfer into a formal setting. For example, the *speech act theory* [Searle, 1969] clearly distinguishes nested modes of human communication, i.e., the *utterance* force of producing some sound, the *locutionary* force of saying some sentence, the *illocutionary* force of meaning some object, and the *perlocutionary* force of causing some effect in the mind of the recipient. Perlocutionary and illocutionary force are particularly different in cases in which the utterer uses an indirect way of persuading the recipient to do something, e.g., by lying.

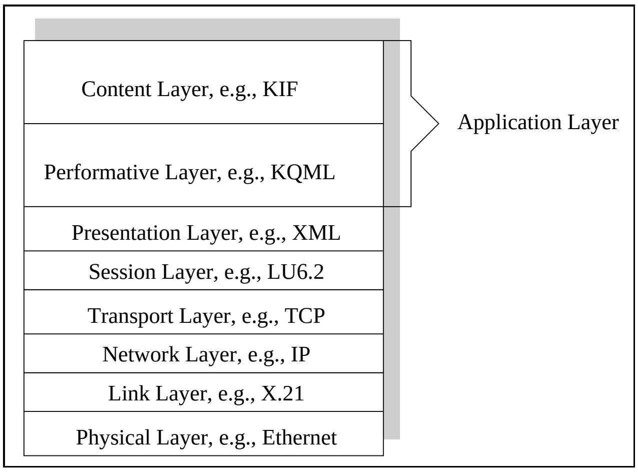

Figure 3: Artificial Communication: KQML and the OSI Reference Model

In a society of benevolent, i.e., truthful, service agents, such as the *Knowledge Query and Manipulation Language* (KQML) [Finin and Fritzson, 1994] presumes, the perlocutionary and illocutionary role of a speech act can be unified. KQML is thus able to identify domain-independent types of speech acts, such as 'telling' or 'requesting' something, which is captured by so-called *performatives*. Languages which address this level of communication are also called *interlinguae*. For example, the *Foundation for Intelligent Physical Agents* (FIPA) [Steiner, 1997] aims to develop an industrial-strength standard quite similar to KQML. Interlinguae are strongly connected to nested *ontolinguae* or *content languages* which are used to represent the domain-dependent *objects* of a performative. Examples of content languages are ISO-*Prolog* [ISO, 1995] or the *Knowledge Interchange Format* (KIF) [Genesereth and et al., 1992].

On the lower level of artificial communication, the human 'producing a sound' is substituted by standardising the information exchange from physical (Ethernet) up to presentational issues (XML, see below). This results in a layered structure (the *Open Systems Interconnection* (OSI) reference model [DIN ISO 7498, 1982]) for KQML communication illustrated by Figure 3. The former OSI application layer now hosts the performative and the content layer. This way, KQML agents which do not share any content language are still able to understand their basic intentions and are thus able to process at least a subset of the utterances.

```
KQML-Content ::= <KQMLCONTENT> (Content|KQML) </KQMLCONTENT>

KQML-Aspect ::= <KQMLASPECT> Content </KQMLASPECT>

Performative ::= "tell" | "deny" |...|
                 "insert" | "delete" |...|
                 "error" | "sorry" | "reply" |...|
                 "evaluate" | "ask-one" | "stream-all"|...|
                 "standby" | "ready" | "next" | "discard" | "eos" |...|
                 "register" | "unregister" | "forward" | "broadcast" |...|
                 "advertise" | "broker-one" |...|

KQML ::= <KQML perf=Performative      language=AttValue
               ontology=AttValue      reply-with=AttValue
               in-reply-to=AttValue   sender=AttValue
               receiver=AttValue      from=AttValue
               to=AttValue            name=AttValue ... >
             KQML-Content KQML-Aspect
         </KQML>
```

Figure 4: Expressing KQML in XML

Syntactically, KQML messages can be encoded using the *eXtensible Markup Language* (XML [Bray, 1997]) as the underlying presentation layer (cf. Figure 4). Originally, KQML uses an ASCII-based string representation. Compliance with todays successful presentation languages, such as the *Hypertext Markup Language* (HTML) [Raggett, 1998], and upcoming standards, like MATHML [Ion, 1998] and OPENMATH [Abbot *et al.*, 1996; Caprotti, 1998], however, is a key issue in designing open systems. These languages use the XML framework as their basis.

For expressing KQML in XML we introduce a special `<KQML/>` tag that is annotated with a particular performative (`perf=`*Performative*). The tag furthermore carries information about the content language used in the KQML message (`language`) and the semantics of nested primitive symbols (`ontology`). The `reply-with` attribute describes whether an answer to the message is expected and with which `in-reply-to` annotation it should be given. The `sender` and `receiver` agents of the message are identified using a unique naming convention, such as *Uniform Resource Locations* (URL's). Sender and receiver can be different from the originator (`from`) and the destination (`to`) of the message. `name` carries the name of some arbitrary agent for introduction purposes.

The actual content of the KQML Message is an expression in the content language *Content* corresponding with the `language` attribution. It is encapsulated in the `<KQMLCONTENT>` tag. Since performatives could be nested, the content could also contain a *KQML* expression itself. The *KQML-Aspect* part of the KQML message specifies whether and with which content the current performative is to be answered.

It is difficult to give a semantics to communication languages in the general case — think of the difference between perlocutionary and illocutionary force. Presuming benevolent agents, however, giving a fixed meaning to each KQML message according to the chosen *Performative* makes sense. Because deliberative capabilities are a necessary precondition for reasonable communication, the identification of an agent with a *virtual* knowledge base (see Figure 2) is helpful for this purpose. Virtual, because not every fact or belief must be present in the state of the knowledge base, but could possibly be deduced from the canonical representation using a rational inference procedure. The semantics of `"tell"`, for example, is straightforward to describe, then: The utterer notifies that the embedded *KQML-Content* is an element of its virtual knowledge base. `"deny"` simply means the contrary.

Indeed, KQML stems from an attempt to combine heterogeneous knowledge sources over the network. The set of performatives and their semantics thus captures all the reasonable interactions between knowledge sources. Besides the *informatives* like `"tell"` and `"deny"`, KQML introduces *database performatives*, such as `"insert"` and `"delete"` with which the utterer suggests the recipient to change the content of its virtual knowledge base. A *basic response* to such a suggestion could be `"error"` (the operation would cause inconsistencies) or `"sorry"` (the recipient is not able to process the operation because of technical reasons, e.g., it is not able or does not have enough computational resources to perform it).

A more sophisticated `"reply"` response is necessary to process the *query* performatives `"evaluate"` and `"ask-one"`. `"evaluate"` requests the recipient just to convert (simplify) the content expression into the canonical representation used by its knowledge base. The simplified expression does not have to be valid for that purpose. By `"ask-one"`, a *match* of the content expression with the virtual knowledge base is invoked, i.e., whether it could be derived from the canonical data. This presupposes the content language to exhibit some notion of partial specification, for example by introducing variables and unification. With respect to the traditional input-output specification of services, talking about constrained objects is a far more expressive scheme. The results of matching are, again, expressions in the content language embedded in a `"reply"` performative. The desired format of responses can be specified in advance by the requesting agent in the *KQML-Aspect* part of the respective query performative.

Access of knowledge sources must not stick with a simple query-response scheme. By allowing for advanced queries with multiple responses (for example, `stream-all`: try to match the content in all possible ways with the virtual knowledge base and send the result in separate `"reply"` messages) and nested performatives, KQML is able to introduce on-demand strategies:

```
<KQML perf="standby">
   <KQMLCONTENT>
      <KQML perf="stream-all">
      <KQMLCONTENT> Match </KQMLCONTENT> </KQML>
   </KQMLCONTENT>
</KQML>
```

This is an exemplary enquiry of a 'client' to a recipient 'server' to prepare for an on-demand streaming service which the server acknowledges with `"ready"`. Now the client is able to utter `"next"` performatives to trigger subsequent replies regarding the *Match* expression. If the service is obsolete, the client sends `"discard"`. If there is no further response, the server sends `"eos"` ('end of stream').

Finally, KQML supports the maintenance of agents' *social model* (Figure 2), i.e., the management of names, characteristics, and capabilities of neighbour agents, in order to build a functional 'neighbourhood' of knowledge sources. `"register"` and `"unregister"` are simple naming capabilities with which agents introduce themselves and exit the society. Thus, agents can maintain a list of active neighbours to which they could `"forward"` embedded KQML messages. The `"broadcast"` performative also uses this mechanism to route KQML messages to all connected agents in a network. The `reply-with` attribution can be used to avoid cycles.

Using nested KQML expressions and the matching principle of the encapsulated content language, the `"advertise"` performative allows to build up a more detailed domain-related model of neighbours. The content of an advertisement are those KQML message patterns which the agent is willing or able to process. Thus each agent is able to maintain a lookup table with agent names and their capabilities in terms of KQML patterns. This table is used, for example, in delegating a particular task to another agent (`"broker-one"`).

Our presentation of KQML performatives has of course neither been exhaustive nor detailed. It should however have become clear that using an agent-oriented architecture and communication language combines the achievements of, e.g., an object-oriented methodology and distributed programming, and is able to provide an open, flexibly interacting, and robust network of software services, such as knowledge bases and mathematical services. As such, agents are **not** a prime construct for improving the expressivity of a service domain which is not the aim of the present article.

4 Agent-Oriented Integration of Mathematical Services

Coincidently, the desiderata for distributed automated theorem proving that we have sketched in Section 2 fit exactly with the application profile of the Agent-Oriented Programming techniques developed in Section 3. Furthermore, a virtual knowledge base agent is very close to a mathematical service: it maintains a set of mathematical 'truths' upon which a rational inference procedure (proof procedure or calculation) operates. For example, a theorem prover virtually represents a knowledge base for all proofs that it could derive. A computer algebra system could be seen as the set of all computations (equations) it could solve. Also the user interface that is able to ask for the user's help represents the combined knowledge of its user. Subsequently, we propose the MATHWEB architecture (Figure 5) as a reasonable, agent-oriented integration of mathematical services.

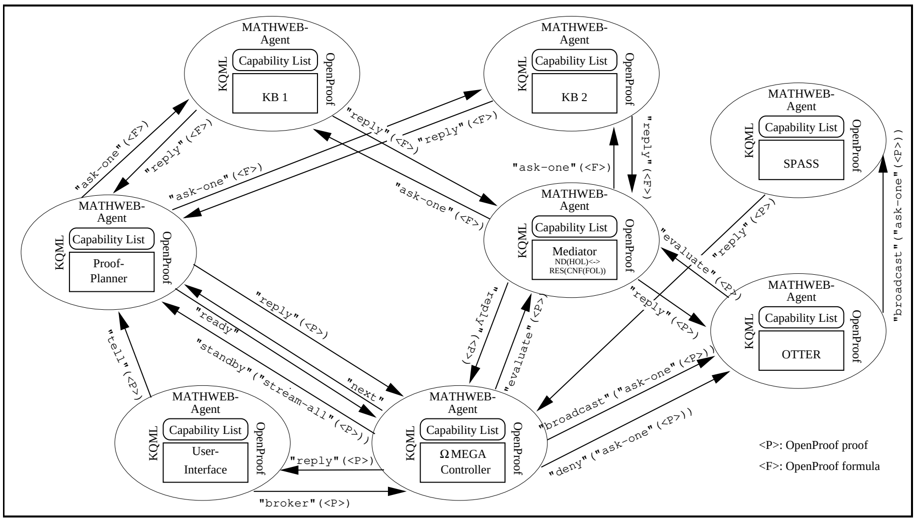

Figure 5: Agents as Distributed Mathematical Services in MATHWEB

Following the Agent-Oriented Programming paradigm, MATHWEB encapsulates mathematical services, such as the user interface, the ΩMEGA control module, the proof planner, knowledge bases, proof mediators, and proof systems like SPASS and OTTER, each into an agent-shell. These agents are reactive in that they are steadily interacting with users and other software agents working on shared proofs and mathematical computations. They are pro-active in that they adopt and autonomously work on particular mathematical goals. And they are social in that they request other agents or even the human user to support the successful execution of their services by communicating via KQML.

MATHWEB embeds a particular content language into KQML. OPENPROOF which is explained in detail in Section 5 is derived from the OPENMATH [Abbot *et al.*, 1996] standard that has been designed as a fundamental (higher-order) language for exchanging mathematical objects, such as symbols, variables, functional abstractions, and applications. OPENPROOF extends this repertoire to represent formulae in various logics, mathematical computations upon those, and especially proofs in different calculi. Using OPENMATH *variables*, these structures can be defined even left partially unspecified which introduces a sophisticated notion of matching a virtual (mathematical) knowledge base: a partial proof or a partial computation can be given in a KQML query. Matching into the virtual knowledge base amounts to deductive or algebraic computations which further instantiate the proof and which will be finally returned in a response performative. Similarly, OPENPROOF expressions can be transformed by the user interface forth-and-back into human-oriented visualisations or verbalisations to interact with the user. In each case, proof and computation structures which are constrained on different levels of representation are fundamentally more powerful and flexible than traditional protocols for specifying deduction problems. For example, the operation of a proof planner, which takes some (partial) proof and returns several (partial) subproblems can be expressed in OpenProof.

It is, however, necessary to care for an efficient treatment of the mathematical structures. MathWeb agents especially assume a clean separation of (meta-)variables for identifying a particular service invocation and (object-)variables in the problem specification. This way, MathWeb agents can pragmatically preprocess received KQML(OpenProof) messages in order to control the encapsulated operation of theorem provers and mathematical systems. Vice versa, it is possible for MathWeb agents to construct KQML(OpenProof) messages if the embedded computations need some support. The generic facility of any MathWeb agent to analyse and generate KQML(OpenProof) is thus coupled to a concrete, service-specific interface. For each type of mathematical services, a suitable interface structure and respective encodings in KQML(OpenProof) can be specified (see Section 5.3 for an overview).

We allow a single agent to manage several, simultaneous instances of the same service, e.g., to elaborate several theorem provers at the same time, but based on shared canonical knowledge (the background theory). This is an important restriction, because the functioning of KQML strongly depends on the constructibility of a consistent virtual knowledge base for each agent. Having several service instances operating on different background theories and allowing different services within the same agent is therefore not advisable.

MathWeb agents maintain a social model of their environment in the form of a capability list, i.e., they keep book about a portion of the overall service architecture. For example, the user interface agent might only know the proof planner and the ΩMEGA control agent. The Spass and Otter agents might only know each other and be aware the mediator agent which encapsulates some of the previous functionality of the ΩMEGA controller to translate between higher-order natural deduction proofs (ND(HOL)) and first-order resolution using clausal normal form (RES(CNF(FOL))). The ΩMEGA control agent could connect the proof planner, theorem provers, and the mediator.

A MathWeb agent organises information about the capabilities of other MathWeb agents in a lookup table. The table stores the incoming `"advertise"` performatives carrying KQML messages to which these agents could successfully respond to (see Section 5.3). Again, the expressiveness of the content language OpenProof is useful to specify, e.g., knowledge bases which are able to deliver formula definitions of mathematical symbols responding to `"ask-one"` messages, proof systems which are able to process similar queries regarding proofs, mediators which could `"evaluate"` formulae or proofs from/into particular formalisms, and even services, such as the proof planner which provide a streaming service in order to transmit multiple partially instantiated (sub-)results on-demand.

Besides the usual `"reply"` performative, answers to service requests in MathWeb will also transport processing errors and technical errors. KQML's `"error"` and `"sorry"` performatives, however, are somehow restricted for this purpose, since ignoring their content. It is thus useful to allow particular error expressions in the content language (Section 5) as ordinary KQML replies. Theorem proving is a challenging domain for coping with failures since it is undecidable in general. How could a service ever return a message saying that a requested proof is not possible? When does a client know that it has received all possible (useful) answers to the requested computation?

It is necessary to take the bounded rationality aspect of MathWeb agents into account right from design time: Objects of mathematical computations should be intimately coupled with the situative context in which they are invoked, i.e., with the resources that they are allowed to consume in processing. For this purpose, OpenProof proofs or computations are annotated with descriptions of the time, the memory, the information, and the user interaction that have been necessary to derive them. This makes it possible to establish *a priori* estimations of the utility of a service which helps to optimise the MathWeb.

Now reconsider the example of Section 2. Figure 5 shows one of the many extended possibilities using MathWeb: Initially, the user interface starts, supervised by a human user, a proof delegation (`"broker-one"`) to the ΩMEGA control agent. We assume that the initial proof goal has been entered by the user. The controller delegates the task of breaking down the proof into proper subproblems to the proof planner agent. All subproblems are requested in the form of a streaming service (`"standby"`). Not until needed in the proof planner, the group definitions referred in the proof specification are looked up by querying the two knowledge base agents. Perhaps with the help of the human user which proposes some instantiation of the proof via `"tell"`, the proof planner constructs appropriate subproblems and replies them to the ΩMEGA controller which, in turn, `"broadcast"`s the higher-order natural deduction structures to the prover agents to concurrently run for solutions. Because Otter and Spass operate on first-order clausal normal form and construct resolution-type proofs, the Otter agent-shell first asks the mediator agent for help in translation (`"evaluate"`) before routing the translated broadcasts to Spass. For this purpose, the mediator agent looks up the remaining definitions, such as of `inverse-exists`, from the knowledge bases. In a simplified version, the sent KQML messages look the following way. The actual content tags use the OpenProof syntax of the following section, of course.

```
<KQML perf="ask-one">
   <KQMLCONTENT>
      F̄ = inverse-exists
   </KQMLCONTENT>
   <KQMLASPECT>
      F̄
   </KQMLASPECT>
</KQML>
```

```
<KQML perf="reply">
   <KQMLCONTENT>
      λG: ι → o. λOP: (ι, ι) → ι. λE: ι.
        ∃ F: ι → ι. ∀ X: ι.
          G(X) → OP(X, F(X)) = E
   </KQMLCONTENT>
</KQML>
```

In our example, Spass has found a result first; the notified controller will then `"deny"` the original request to shut down the redundant computations in Otter. Finally, when all subproofs are collected by the controller, the mediator agent is once again contacted to transform the overall result back into natural deduction form which is used in the user interface for presentation purposes.

With respect to the central role of the ΩMEGA controller in the heart of Figure 1, MathWeb now amounts to a dynamically rearrangeable decentralisation. This has been possible due to the richness of agent-based communication. Especially we can now uncouple the syntax translations necessary to communicate subproblems and proofs between the ΩMEGA controller and the theorem proving agents. The unified view onto (mathematical) services allows to integrate further services without ubiquitous re-engineering of proprietary interfaces. Instead, the envisaged application is wrapped into the generic MathWeb shell by customising a library of suitable interfaces. A further advantage of such an open approach is that several users with different demands can use the system cooperatively or independently at the same time. The particular modules then decide based on priority and workload whether to process particular tasks or not. In short, MathWeb provides the modular, inter-operable, robust, and scalable framework for automated theorem proving motivated by this article. Of course, MathWeb owes much to the OpenProof content language that we outline in the following section.

5 A Content Language for Mathematics and Deduction

Given a generic interlingua such as KQML, it additionally requires a suitable ontolingua to express the content of services to talk about service performances a particular application domain for interacting agents. In the case of mathematical theorem proving, this content comprises mathematical objects, formulae, theorems, theories, but also (partial) proofs, and even proof plans. Appropriate candidates for such a language are the so-called 'DFG syntax' [Hähnle *et al.*, 1996] or the specification put forward by the OpenMath initiative (see `http://www.openmath.org`), which strives for a standard exchange platform for mathematical software systems. For MathWeb, we propose a content language (see 5.1 for details) which is an extension of the latter, since it has more support for distribution and also covers symbolic computation services. There are even already some OpenMath-compliant systems, such as Maple [Redfern, 1998], which can immediately serve as mathematical services.

We will now give a brief overview on the emerging OpenMath standard (cf. [Caprotti, 1998]) and indicate where it meets the communication needs for MathWeb. In the Section 5.2 we will extend the OpenMath suiting our needs to a content language, which in lack of a better name we have called OpenProof.

5.1 The OpenMath Standard

The OpenMath initiative's aim is to establish a common information exchange platform among software tools used in mathematics. At the moment, their efforts are largely focusing at representational issues for the communication between computer algebra systems. We will use the mechanisms provided by the OpenMath standard to express the logical side of mathematics (definitions, theorems,...), too.

The OpenMath language is syntactically a member of the XML [Bray, 1997] family of languages to which also HTML [Raggett, 1998] or its extension for mathematics, MathML [Ion, 1998], belong. XML derivates can be nested, thus OpenMath expressions fit very well into our KQML variant of Section 3. The OpenMath standard defines a canonical way to represent the structure of mathematical objects. It offers primitive constructs for logical constants (called 'symbols' in OpenMath and indicated by the `<OMS/>` tag), variables (`<OMV/>`), applications (by `<OMA/>`), and a primitive binding construct `<OMBIND/>` that allows to formalise quantifiers or λ-abstraction (the bound variables are tagged using

<OMBVAR/>). For instance the expression $\sin(x)$ and the function $f(x) = \sin(x)$ have the following OpenMath representations

```
<OMOBJ><OMA>
   <OMS cd="trig" name="sin"/>
   <OMV name="x"/>
</OMA> </OMOBJ>
```

```
<OMOBJ><OMBIND>
   <OMS cd="ecc" name="Lambda"/>
   <OMBVAR>
      <OMV name="x"/>
   </OMBVAR>
   <OMA>
      <OMS cd="trig" name="sin"/>
      <OMV name="x"/>
   </OMA>
</OMBIND></OMOBJ>
```

In order to support a standardised semantics, especially when resolving symbols in OpenMath syntax, a set of so-called *content-dictionaries*, referred by the cd attribute of OpenMath symbols is provided. Content dictionaries are globally agreed on specifications on the meaning of OpenMath symbols. Based on uniquely named content dictionaries, the individual mathematical systems implement so-called *phrase-books*, i.e., transformation procedures that interpret OpenMath representations and transform them into internal representations of the systems proper (and vice versa). Therefore, such phrase books are an integral part of the interface between a mathematical service and the embracing MathWeb agent. Note that due to the explicit annotation of individual symbols, the original ontology attribute in KQML-performatives which is a more rigid way of fixing the semantics of symbols becomes redundant.

There are some special tags for grounding integers (<OMI>), floats (<OMF>, strings (<OMSTR>), and byte arrays (<OMB>) directly in the language. Furthermore, the OpenMath protocol provides so-called 'error objects' that allow to pass information about exceptional computation states in the mathematical services themselves. Errors are OpenMath symbols applied to a list of objects. Consider for instance the following representation of division by zero:

```
<OMOBJ><OME>
   <OMS cd="arith" name="DivisionByZero"/>
   <OMA>
      <OMS cd="arith" name="divide"/>
      <OMV name="x"/>
      <OMI> 0 </OMI>
   </OMA>
</OME></OMOBJ>
```

KQML usually manages failure handling by its builtin performatives, e.g., "error" and "sorry", annotated with some comment or code of the failure. In MathWeb, this would amount to an extensive list of failure codes depending on the various mathematical services. A "reply" containing an OpenMath error object is however more informative both on the mathematical and the deductive service level.

5.2 OpenProof: Formulae and Proofs in OpenMath

OpenMath is simplistic in that it does not immediately introduce logical expressions, e.g., from propositional logic, equality logic, clause logic, higher-order logic, etc. — let alone proofs in various calculi, such as first-order natural deduction or higher-order semantic tableaux. Similarly, there is no notion of a mathematical computation including intermediate results.

OpenMath's expressive binding constructor, however, allows us to build such structures as mathematical objects including 'meta'-information, e.g., in which logical language a formula is expressed, and 'meta'-variables, e.g., unspecified parts of a logical formula, using a new OpenProof content dictionary and additional dictionaries for particular logics and proof calculi. We will elaborate on this approach in the rest of the section without giving a formal definition of the `openproof` dictionary, which is outside the scope of this article. To conserve space, we will replace some lengthy syntactical forms with conventional mathematical notation for presentation purposes.

The `openproof` content dictionary introduces four new binding symbols: `formula` and `term` (for formulae and terms containing meta-variables), `proof` and `computation` (for proofs and computation objects) in OpenMath. Attributions of variables allow us to make assertions about the type and syntactical nature of the logical objects they represent; this will become essential for describing the effect of mathematical services. Consider for instance the following OpenMath representation.

```
<OMOBJ><OMBIND>
   <OMS cd="openproof" name="formula"/>
   <OMBVAR>
      <OMATTR><OMATP>
                <OMS cd="openproof" name="language"/>
                <OMS cd="FFOL" name="CNF"/>
              </OMATP>
         <OMV name="F"/>
      </OMATTR>
   </OMBVAR>
   <OMV name="F"/>
</OMBIND><OMOBJ>
```

It stands for any formula F that is a first-order formula in conjunctive normal form. Here we assume the existence of a content dictionary `FFOL` 'Fragments of first-order logic' that defines first-order logic (i.e. the logical symbols $\forall, \exists, \wedge, \vee, \neg \ldots$) and various sub-languages. Along the same lines, we represent the schematic term $X + X$, where the meta-variable X stands for an arithmetic expression (as defined in the content dictionary `arith`):

```
<OMOBJ><OMBIND>
   <OMS cd="openproof" name="term"/>
   <OMBVAR><OMATTR>
      <OMATP><OMS cd="arith" name="arith-expression"/></OMATP>
      <OMV name="X"/>
   </OMATTR></OMBVAR>
   <OMA>
      <OMS cd="arith" name="plus"/>
      <OMV name="X"/>
      <OMV name="X"/>
   </OMA>
</OMBIND><OMOBJ>
```

OpenProof representations for proofs and computation objects are defined in much the same way. Conceptually, they are five-tuples $(dec, obj, seq, res, lang)$, where

1. *dec* is a set of declarations for meta-variables in the proof or computation object.
2. *obj* is the proof object or the computation object itself, i.e. a tree representation of the proof or the computation (see [Homann and Calmet, 1996] for details).
3. For a proof object, *seq* is a sequent $\mathcal{H} \vdash A$, where $\mathcal{H}$ is the set of hypotheses, and A is the assertion of *obj*; for a computation object, *seq* is a computation sequent, $\mathcal{A} \mapsto R$, where $\mathcal{A}$ is a set of argument objects and R is the resulting object of the computation.
4. *res* is a specification of the resources used by the proof or computation object. We have already motivated that such an annotation is essential for providing effective mathematical services.
5. *lang* is the logical system that is used to represent the meta-formulae.

These five-tuples are represented as binding objects `proof` and `computation`, where *dec* is represented as the list of attributed bound variables and *seq*, *res*, and *lang* are represented as attributions to *obj*. It is straightforward to use OpenMath terms to encode formal proofs using ideas from the so-called 'propositions-as-types' paradigm (or the *Curry-Howard isomorphism* [Thompson, 1991]). For instance, the λ-term $\Rightarrow I(\lambda X_{A\wedge B}. \wedge I(\wedge ER(X), \wedge EL(X))$ is a representation of the following Natural Deduction proof with its attached OpenMath representation. For simplifying uncritical parts of the lengthy expression, we use convenient conventional notations:

$$\dfrac{\dfrac{\dfrac{[A\wedge B]^1}{B}\wedge ER \quad \dfrac{[A\wedge B]^1}{A}\wedge EL}{B\wedge A}\wedge I}{A\wedge B \Rightarrow B\wedge A}\Rightarrow I^1$$

```
<OMOBJ><OMBIND><OMS cd="ND(FOL)" name="impliesI"/>
   <OMBVAR><OMATTR>
      <OMATP><OMS cd="openproof" name="assertion"/>
         A ∧ B
      </OMATP>
   <OMV name="X"/>
</OMATTR></OMBVAR>
   <OMA><OMS cd="ND(FOL)" name="andI" >
     <OMA><OMA><OMS cd="ND(FOL)" name="andEr">
             <OMV name="X"/>
           </OMA>
        <OMA><OMS cd="ND(FOL)" name="andEl">
           <OMV name="X"/>
   </OMA></OMA></OMA>
</OMBIND></OMOBJ>
```

Here, we assume the existence of a content dictionary `ND(FOL)`, which specifies a Natural Deduction calculus for first-order logic [Gentzen, 1935] by defining the inference rules as OpenMath symbols `impliesI`, `impliesE`, `andI`, `andE`,... (of appropriate types). Note that in contrast to the classical 'propositions-as-types' approach, we have made use of the OpenMath binding construct again to eliminate the λ-abstraction in the argument of $\Rightarrow I$, instead we have made the symbol `impliesI` a binding symbol itself. This is unnecessary from a theoretical point of view, but gives a more direct encoding of the respective proof trees. Now, we can express partially specified proof objects by introducing meta-variables.

$$\cfrac{\cfrac{\begin{array}{cc}[A\wedge B]^1 & \cfrac{[A\wedge B]^1}{A}\,\wedge EL\\ \vdots & \end{array}}{B\wedge A}\,\wedge I}{A\wedge B\Rightarrow B\wedge A}\,\Rightarrow I^1$$

```
<OMOBJ><OMBIND><OMS cd="openproof" name "proof"/>
   <OMBVAR><OMATTR>
      <OMATP><OMS cd="openproof" name="sequent"/>
         A ∧ B ⊢ A
         <OMS cd="openproof" name="language"/>
         <OMS cd="ND(FOL)" name="FO-ND"/></OMATP>
      <OMV name="F"/>
   </OMATTR></OMBVAR>
   <OMATTR><OMATP><OMS cd="openproof" name="sequent"/>
                  ∅ ⊢ A ∧ B ⇒ B ∧ A
           <OMS cd="openproof" name="resources"/>
            3 ≤ R̄ ≤  5
           <OMS cd="openproof" name="language"/>
           <OMS cd="ND(FOL)" name="FO-ND"/></OMATP>
        ⇒ I(λX. ∧ I(∧EL(X),F(X))
   </OMATTR>
</OMBIND> </OMOBJ>
```

In this partial proof, the meta-variable F stands for a sub-proof in first-order ND for the sequent $A \wedge B \vdash B$; F is bound in the `proof` environment and the information about the calculus and the sequent are added by attribution. The resources $\overline{R}$ used by the overall proof are at least 3 ND proof steps and should not exceed 5 proof steps. The partial proof above could be sent to a MathWeb theorem proving agent using the KQML-performative `"ask-one"`: the sending agent wants to know whether there is a single instance of this proof (given the resource bounds of 5 steps) in the virtual knowledge base of the prover agent. The answer could be the OpenProof equivalent to $\Rightarrow I(\lambda X_{A\wedge B}. \wedge I(\wedge ER(X), \wedge EL(X))$ (see above) which has the variable F instantiated by the (functional) symbol $\wedge ER$ and carries the final resource amount of 4 steps. We will come back to the issue of dealing with resources in the conclusion (Section 7).

Note that the flexibility of KQML communication based on meta-variables has to be paid with the necessity of requiring matching the level of agents. In this respect, the content-language OpenProof (and for the same reason already OpenMath) is more problematic than traditional agent content languages. However, since we can restrict OpenProof to second-order expressions, we only need second-order matching, which is known to be decidable [Huet and Lang, 1978]. We are currently investigating whether more restrictive policies for addressing services via OpenProof can be captured with comparably more lightweight mechanisms.

5.3 A Categorisation of Mathematical Services

In this section we briefly categorise mathematical services by their behaviour and communication needs. A special emphasis is put on specifying possible interactions with other agents in MathWeb, thus on the suitability of messages in KQML(OpenProof). We follow categorisations made in [Homann and Calmet, 1996; Hess *et al.*, 1998] and do not claim that our list is complete.

5.3.1 Mathematical Filters

Certain mathematical programs can be used in a filter-like way, that is they can read a request from an input stream and write some answer to an output stream. Mathematical filters can be further grouped in *computation filters* and *deduction filters*. The first perform some numerical or algebraic calculation which result they return (maybe coupled with some protocol information on how the result was obtained), while the latter attempt to prove a given problem and return, if successful, the proof or signal failure. Using partially specified KQML(OpenProof) proof and computation expressions, mathematical filters can be genuinely addressed in one-solution, single-shot modes up to all-solutions, streaming modes.

Unlike computation filters which terminate eventually, deduction filters will not always return a result. Thus deduction agents need to have additional properties for maintenance: On the one hand a requesting client must be able to send a termination signal, e.g., `"deny"`, to a deduction service in order to declare an earlier request as obsolete. On the other hand the service itself needs to survey its own running processes, assign resources to incoming requests and terminate processes that have not produced any results after their allocated resources have been consumed. Two instances of filter agents that we have already integrated into Ωmega are the automatic theorem prover Spass [Weidenbach, 1997] and the Computer Algebra system Maple [Redfern, 1998]. Furthermore, there is a service `competitive-atp` that calls sets of ATP concurrently as competing services (this strategy is known to yield even super-linear speedups in practice).

5.3.2 Mediators

Although OpenProof is a generic representation device for formulae in various logics and proofs in different calculi, it would be an overkill to demand from each MathWeb agent to cope with arbitrary structures besides the natural format of its encapsulated service. This would increase the computational burden that the agent shell has to carry. Instead, the problems involved in translating between the different formats are rather themselves reasonable mathematical services (see the example of Figure 5) to be embedded into agents and to be integrated into the MathWeb. An example of such a mediator agent is a syntax transformer that can convert between different representations of first-order logic, e.g., negation normal form and clausal form. Another service is 'relativisation' which transforms formulae of sorted first-order logic or higher-order logic to classical first-order logic [Schmidt-Schauß, 1989; Kerber, 1991]. Finally, there are proof transformers [Pfenning, 1987; Huang and Fiedler, 1996] that can transform from one calculus into another one (possibly even transforming the base logic along the way). Since mediators do not need a virtual knowledge base in the KQMLsense for that purpose, we rather regard their task to simplify incoming expressions into a canonical format, thus implement the `"evaluate"` performative of KQML with corresponding OpenProof formula or proof contents.

5.3.3 Knowledge Bases

Mathematical knowledge bases are used to uniquely store formulae (axiomatisations, definitions, etc.) and also proof steps and proofs in order to give commonly used, convenient symbols a semantics. Thus, they are a similar concept to the OpenMath content dictionaries. For MathWeb, a close connection of these concepts is envisaged: Knowledge bases with a MathWeb shell are automatically able to produce proper OpenMath code of their knowledge, thus a reasonable content dictionary. On the other hand, MathWeb knowledge bases can access existing content dictionaries to provide their information in the MathWeb.

This happens typically over the "ask-one" performative carrying a higher-order equation (see Section 4).

Contrary to other mathematical services, knowledge bases have the property that they can be dynamically changed by clients, i.e., the user edits a definition in the user interface and "insert"s it to the knowledge base or requests a "delete". The knowledge base agents therefore have some additional information on access rights for particular agents/particular users sending requests. The Mizar Library [Rudnicki, 1992] is a knowledge base that already offers its services via the Internet, but is not yet integrated into MathWeb.

MathWeb currently only includes the MBase service, a simple web-based mathematical knowledge base system that stores mathematical facts like theorems, definitions and proofs and can perform type checking, definition expansion and semantic search. It communicates with other mathematical services by mediators and with humans by the interaction unit OctOpus.

5.3.4 Display Components

This point covers possible interaction devices that enable a user to view and elaborate processed mathematical data in a desired way. To these services belong (graphical or non-graphical) displays and browsers for formulas and proofs, as well as systems that can transform provided data into a human-oriented format. As an example of the latter, one might consider systems that translate proofs into natural language. An example for a graphical user interface that is already available in MathWeb is $\mathcal{L}\Omega\mathcal{UI}$ [Siekmann *et al.*, 1998] the interface for the Ωmega system. The user interface is a basic source of activity, as maintenance of knowledge bases, transformations of logical expressions, and the initiation of proofs are triggered from here (see the above services). Note that the user, thus his user interface, could also appear as requestable entity in the MathWeb, for example to propose an instantiation, to solve some lemma, etc. Therefore, the user interface should also process incoming "ask-one" commands, but for upholding the convenience of the user, it should not accept "stream-all" or similar requests.

5.3.5 Anytime Services

Anytime services provide a means to organise the output of computations that might have more than one result (possibly infinitely many results) to a clients request. The general functionality of these services is similar to those of mathematical filters, but they can also store additional information on both the requested service and the client. The latter itself receives a result along with information on how long the anytime service can provide further results and how these results can be retrieved. Using these specifications, subsequent requests of the client can then be answered by the anytime service using the already computed results (always provided the requests are within the given time limit). A predestined candidate for an anytime agent is, for instance, a unification engine for higher order logics. The necessary information exchange can be encoded in KQML using streaming together with the resource attribution of OpenProof.

5.3.6 Mathematical Control Units

Finally control units form the link between several different other mathematical services. They have the ability to permanently store data of ongoing proofs or computations, making it available to other agents requests as well as using it to assign other agents to certain tasks. While incorporating less or no application services, control units overview a greater portion of the overall service architecture and function as brokers to which agents with a smaller 'social horizon' could turn to. MATHWEB should always have a 'backbone' of persistent, mutually-aware control units in order to bridge dispersed areas of services. As elaborated in the example in Section 4 the ΩMEGA control unit contains all necessary information in order to carry out the steps leading to a complete proof of the given example. `"broadcast"`, `"forward"`, and `"broker-one"` messages are typically sent to the control unit for routing purposes.

6 Implementation and Experiences

MATHWEB is implemented as an object-oriented toolbox that provides the functionality for building a society of software agents that render mathematical services by either encapsulating legacy deduction software or their own functionality. The system is available at `http://www.ags.uni-sb.de/~omega/www/mathweb.html`.

The current list of integrated mathematical services consist of the theorem provers and computer algebra systems mentioned in the introduction, the knowledge base system MBASE, the proof transformation and presentation system PROVERB [Huang and Fiedler, 1996] and the $\mathcal{L}\Omega\mathcal{UI}$ [Siekmann *et al.*, 1998] and OCTOPUS user interfaces. Currently, these services are used by the three control components INKA [Hutter and Sengler, 1996], λ*Clam* [Richardson *et al.*, 1998], and the ΩMEGA kernel [Benzmüller *et al.*, 1997]. A first synergy effect of MATHWEB has been that the first two systems can now partake in infrastructure (such as $\mathcal{L}\Omega\mathcal{UI}$ and MBASE) developed for the latter, while the ΩMEGA system can now turn to INKA or λ*Clam* when it needs support for inductive proofs. Furthermore, MATHWEB approach has been a key factor in keeping the ΩMEGA system maintainable and the near future will see further modularisation and 'agentification' of system components, which will lead to simpler system maintenance and a more open development model.

In the current implementation, the software bus functionality of MATHWEB is realized by a CORBA-like model [Siegel, 1996] in which a central *broker* agent provides routing and authentication information to the mathematical services (see [Hess *et al.*, 1998] for details). The agents are realized in a distributed programming system MOZART (see `http://www.mozart-oz.org` for details), an interactive and distributed implementation of the concurrent constraint programming language OZ [Smolka, 1995]. MOZART

Furthermore, MATHWEB provides the MOZART shell (MOSH), a tool for launching and administering multiple MOZART applications (the agents) within only one MOZART process. It combines some frequently used shell commands (for files, processes and environment) with some (thread-related) MOZART commands. These allow (remotely) administering the mathematical services across the Internet, since the administrator can connect to remote MOSH daemons –

which run continually at the host providing the services – launch and terminate services. This also allows for a limited form of self-organization of mathematical services, since these can use MOSH scripts themselves to launch and administer other services.

By providing several trading points which are interconnected using the KQML interlingua we are now smoothly migrating into the fully distributed MATHWEB in which each mathematical service agent possesses the complete functionality of the trading point including the capability lookup table. The current trading points still use a proprietary protocol — both at the interlingua and the content language levels — for communication with several embedded mathematical services that is customised to the current needs and functionality of ΩMEGA. Since it resembles KQML performatives, we will come up with a fully KQML-compliant system in the near future.

mOZART's main advantage as a basis for MATHWEB comes from its network transparency, i.e., the full support of remote computations in the base language (lexical scoping, logical variables, objects, constraints,...), and its network awareness, i.e., the full control over network operations, such as the choice between stationary and mobile objects, which make it easy to 'agentify' arbitrary applications. Incorporating both properties goes well beyond the distribution facilities of e.g. CORBA. mOZART also provides high-level inference primitives like constraint propagation, search, and search control which makes it a good implementation choice for the mathematical services proper. mOZART provides low-level primitives to seamlessly integrate C/C++ code and to control arbitrary external processes via Operating System functionality. An example of a mathematical service that is fully implemented in mOZART is the generic proof visualisation tool $\mathcal{L}\Omega\mathcal{UI}$.

For the content language, MATHWEB still uses a mix of languages, consisting mainly of the ΩMEGA, INKA, and *λClam* internal formats and the various input languages of the legacy systems, turned into mathematical services by MATHWEB. Work is under way to implement the translation services needed for integrating the content language OPENPROOF proposed in section 5.2. A unified content language will greatly simplify the administration of mathematical services, since with n input languages of legacy systems, we only need $2n$ transformation services for pairwise communication instead of n^2 without OPENPROOF. In fact, the need for a development content language came from this practical need as much as the desire for standardisation.

Apart from the application in mathematics and software engineering that has been the primary focus, MATHWEB has been tested in the DORIS[1] system, a natural language understanding system that uses first-order automated theorem provers and model builders as external mathematical services to solve the consistency and entailment problems pertaining to various disambiguation problems in text and dialogue understanding. DORIS generates up to 500 deduction problems for each sentence it processes, distributes them to competing mathematical services (over a network of workstations) and collects the results to obtain the desired result. Using the MATHWEB approach, the integration of the theorem provers was very simple: the only new parts were a socket connection from Pro-

[1] See `http://www.coli.uni-sb.de/~bos/atp/doris.html` for a web-based interface that acts as a MATHWEB client.

log on the DORIS side and a new service module for the `doris` service[2] on the MATHWEB side. Experience with this application shows that distribution using MATHWEB does not come for free: A test with around 1300 DORIS deduction queries yielded the following timings:[3]

30–1250 ms pure theorem proving time
50-120 ms spent in the service module (opening an inferior shell, creating files,...). This depends strongly on the efficiency of the server file system.
5–500 ms Internet latency (we have measured inter-departmental (in Saarbrücken) and international (Saarbrücken/Amsterdam) connections)

However, the large number of deduction problems and the possibility of coarse-grained parallelisation by distribution lead to a significant increase in overall system performance, compared to an earlier centralised, sequential architecture. In particular, the timings also show that it can pay off for a client in Saarbrücken to delegate deduction problems to faster machines in Amsterdam or vice versa.

7 Conclusion

We have proposed a distributed network architecture for automated and interactive theorem proving, MATHWEB, that extends and generalises earlier efforts in the ΩMEGA proof development system to support modularisation, interoperability, robustness, and scalability of mathematical software systems. The key concept is the identification of mathematical applications with communicating, autonomous agents, called mathematical services. We have described an agent– and communication model for the MATHWEB architecture based on the KQML and OPENMATH standards which provides the functionality to turn existing theorem proving systems and tools into mathematical services homogeneously integrated into a networked proof development environment.

7.1 Resources

Future work will concentrate on the resource part of OPENPROOF, since the number of proof steps used in the examples in section 5.2 is certainly not an universally meaningful unit of measure throughout the MATHWEB. Particular provers and calculi need less basic operations than others for performing particular manipulations or inferences. Particular machines running MATHWEB agents are faster than others. Thus having comparable processing times seems to be a better approach. Additional costs, such as memory usage, required transformations, the information looked up in knowledge bases, user interactions, etc., are also not yet accounted for. Gerber & Jung [Gerber and Jung, 1998] propose *abstract resources* as a reasonable representation device for such interdependencies between autonomous agents. They furthermore describe topological and algorithmic means for organising a society of agents towards optimality based on abstract resources.

[2] I.e. a small (60 line) MOZART program that relays problems, results and statistics between the DORIS program and the `competitive-atp` service.

[3] These times have been measured on a collection of SUN Ultra machines running Solaris 5 in Saarbrücken and Amsterdam (all timings given in total elapsed time; normalised to our fastest machine, a SUN Ultra 4 at 300 MHz).

7.2 Negotiation

One means for load-balancing in multi-agent systems with central decision making has been adopted from economics: the market metaphor. If (mathematical) agents are equipped with a notion of (selfish) utility, thus money, and is provided a communicative platform for performing negotiations, the whole system is able to perform self-regulation, i.e., suboptimalities from inefficient or disabled services will be adapted by reorientation of service requests. The contract net protocol [Smith, 1980] and its derivates, for example, introduce an auction mechanism for the delegation of tasks charged with certain costs. An agent herein 'announces' a task, such as the proof of a certain theorem, to a number of service agents. Each service now judges its competence and predicts the expected costs that his processing will produce. It 'bids' for the task accordingly. The initiative agent then selects one or several service agents in order to reduce costs and maximise performance. Using the KQML performatives, such auction mechanisms can be easily implemented.

Acknowledgements

The work reported here was supported by the 'Deutsche Forschungsgemeinschaft' (DFG) in the 'Sonderforschungsbereich 378' and the 'Graduiertenkolleg Kognitionswissenschaft'. The authors would like to thank the ΩMEGA group at the Universität des Saarlandes for valuable discussions. Furthermore, we would like to thank the Oz development team at the Universität des Saarlandes. Without the mOZart programming language and the readily shared know-how about it, the whole enterprise of distributing ΩMEGA and developing MathWeb would have been impossible.

References

[Abbot *et al.*, 1996] J. Abbot, A. v. Leeuwen, and A. Strotmann. Objectives of OpenMath. Technical Report 12, RIACA, Technische Universiteit Eindhoven, 1996.

[Andrews *et al.*, 1996] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[Andrews, 1986] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.

[Ballarin *et al.*, 1995] C. Ballarin, K. Homann, and J. Calmet. Theorems and Algorithms: An Interface between Isabelle and Maple. In A. Levelt, editor, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, pp. 150–157. ACM Press, 1995.

[Benzmüller and Kohlhase, 1998] C. Benzmüller and M. Kohlhase. LEO – a Higher Order Theorem Prover. In Kirchner and Kirchner [1998], pp. 139–144.

[Benzmüller and Sorge, 1998] C. Benzmüller and V. Sorge. Integrating Tps with ΩMEGA. In J. Grundy and M. Newey, editors, *Theorem Proving in Higher Order Logics: Emerging Trends*, Technical Report 98-08, Department of Computer Science, pp. 1–19, Canberra, Australia, 1998. The Australian National University. available from `http://cs.anu.edu.au/techreports/recent.html`.

[Benzmüller *et al.*, 1997] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a Mathematical Assistant. In McCune [1997], pp. 252–255.

[Boyer and Moore, 1979] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, New York, 1979.

[Bray, 1997] Extensible Markup Language (XML). W3C Recommendation PR-xml-971208, W3 Consortium, 1997. Available at `http://www.w3.org/TR/PR-xml.html`.

[Calmet and Homann, 1997] J. Calmet and K. Homann. Towards the Mathematical Software Bus. *Journal of Theoretical Computer Science*, 187(1–2):221–230, 1997.

[Caprotti, 1998] Draft of the open math standard. Open Math Consortium, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.

[Cheikhrouhou and Sorge, 1998] L. Cheikhrouhou and V. Sorge. Planning Equivalence Proofs. In Denzinger et al. [1998], pp. 13–22.

[Clarke and Zhao, 1992] E. Clarke and X. Zhao. Analytica-A Theorem Prover in Mathematica. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, pp. 761–763. Springer, Berlin, 1992.

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

[Dahn, 1997] I. Dahn. Integration of Automated and Interactive Theorem Proving in ILF. In McCune [1997], pp. 57–60.

[Denzinger and Dahn, 1998] Jörg Denzinger and Ingo Dahn. Cooperating theorem provers. In Wolfgang Bibel and Peter Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume 2, pp. 483–416. Kluwer, 1998.

[Denzinger *et al.*, 1997] J. Denzinger, J. Kronenburg, and S. Schulz. DISCOUNT - A distributed and learning equational prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[Denzinger *et al.*, 1998] J. Denzinger, M. Kohlhase, and B. Spencer, editors. *CADE-15 Workshop "Using AI Methods in Deduction"*, 1998.

[Denzinger, 1993] J. Denzinger. *Teamwork: A method to design distributed knowledge based theorem provers*. PhD thesis, Universität Kaiserslautern, 1993. (in german).

[DIN ISO 7498, 1982] Informationsverarbeitung DIN ISO 7498. Kommunikation Offener Systeme, Basis-Referenzmodell, 1982.

[Felty and Howe, 1997] A. P. Felty and D. J. Howe. Hybrid Interactive Theorem Proving Using Nuprl and HOL. In McCune [1997], pp. 351–365.

[Finin and Fritzson, 1994] T. Finin and R. Fritzson. KQML — A Language and Protocol for Knowledge and Information Exchange. In *Proceedings of the 13th Intl. Distributed AI Workshop*, pp. 127–136, Seattle, 1994.

[Fisher and Ireland, 1998] M. Fisher and A. Ireland. Multi-Agent Proof-Planning. In Denzinger et al. [1998], pp. 33–42.

[Fisher, 1997] M. Fisher. An Open Approach to Concurrent Theorem Proving. In J. Geller, H. Kitano, and C. Suttner, editors, *Parallel Processing for Artificial Intelligence*, volume 3. Elsevier/North Holland, 1997.

[Genesereth and et al., 1992] M. Genesereth and R. Fikes et al. Knowledge Interchange Format: Version 3.0 Reference Manual. Technical report, Computer Science Department, Stanford University, 1992.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, 39:176–210, 572–595, 1935.

[Gerber and Jung, 1998] C. Gerber and C. G. Jung. Resource management for boundedly optimal agent societies. In *Proceedings of the ECAI´98 Workshop on Monitoring and Control of Real-Time Intelligent Systems*, pp. 23–28, 1998.

[Giunchiglia *et al.*, 1996] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories – Towards an Architecture for Open Mechanized Reasoning Systems. In F. Baader and K. Schulz, editors, *Frontiers of combining systems*, volume 3 of *Applied logic series*, pp. 157–174. Kluwer Academic Publishers, Dordrecht, 1996.

[Good, 1976] I. J. Good. *Good Thinking*. University of Minnesota Press, Minneapolis, 1976.

[Hähnle *et al.*, 1996] R. Hähnle, M. Kerber, and C. Weidenbach. Common Syntax of DFG-Schwerpunktprogramm "Deduktion". Internal Report 10/96, Universität Karlsruhe, Fakultät für Informatik, 1996.

[Hall, 1959] M. Hall. *The Theory of Groups.* The Macmillan Company, New York, 1959.

[Harrison and Théry, 1993] J. Harrison and L. Théry. Reasoning About the Reals: The Marriage of HOL and Maple. In A. Voronkov, editor, *Proceedings of the 4th International Conference on Logic Programming and Automated Reasoning (LPAR'93)*, volume 698 of *LNAI*, pp. 351–353. Springer, Berlin, 1993.

[Hess *et al.*, 1998] S. Hess, Ch. Jung, M. Kohlhase, and V. Sorge. An Implementation of Distributed Mathematical Services. In *6th CALCULEMUS and TYPES Workshop*, Eindhoven, 1998. Electronic Proceedings `http://www.win.tue.nl/math/dw/pp/calc/proceedings.html`.

[Homann and Calmet, 1995] K. Homann and J. Calmet. An Open Environment for Doing Mathematics. In M. Wester, S. Steinberg, and M. Jahn, editors, *Proceedings of 1st International IMACS Conference on Applications of Computer Algebra*, Albuquerque, 1995.

[Homann and Calmet, 1996] K. Homann and J. Calmet. Structures for Symbolic Mathematical Reasoning and Computation. In J. Calmet and C. Limogelli, editors, *Design and Implementation of Symbolic Computation Systems, DISCO'96*, no. 1128 in LNCS, pp. 216–227. Springer, Berlin, 1996.

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

[Huang and Fiedler, 1996] X. Huang and A. Fiedler. Presenting Machine-Found Proofs. In McRobbie and Slaney [1996], pp. 221–225.

[Huet and Lang, 1978] G. Huet and B. Lang. Proving and applying Program Transformations expressed with Second Order Logic. *Acta Informatica*, 11:31–55, 1978.

[Hutter and Sengler, 1996] Dieter Hutter and Claus Sengler. INKA - The Next Generation. In McRobbie and Slaney [1996], pp. 288–292.

[Hutter *et al.*, 1996] D. Hutter, B. Langenstein, C. Sengler, J. Siekmann, W. Stephan, and A. Wolpers. Verification Support Environment. *High Integrity Systems*, 1(6), 1996.

[Ion, 1998] Mathematical Markup Language (MathML) 1.0 specification. W3C Recommendation REC-MathML-19980407, W3 Consortium, 1998. Available at `http://www.w3.org/TR/REC-MathML/`.

[ISO, 1995] Information technology – Programming languages – Prolog – Part 1: General core, 1995.

[Kerber *et al.*, 1998] M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[Kerber, 1991] M. Kerber. How to Prove Higher Order Theorems in First Order Logic. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 137–142. Morgan Kaufmann, San Mateo, 1991.

[Kirchner and Kirchner, 1998] C. Kirchner and H. Kirchner, editors. *Proceedings of the 15th Conference on Automated Deduction*, no. 1421 in LNAI. Springer, Berlin, 1998.

[McCune and Wos, 1997] W. McCune and L. Wos. Otter CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.

[McCune, 1997] W. McCune, editor. *Proceedings of the 14th Conference on Automated Deduction*, no. 1249 in LNAI. Springer, Berlin, 1997.

[McRobbie and Slaney, 1996] M. McRobbie and J. Slaney, editors. *Proceedings of the 13th Conference on Automated Deduction*, no. 1104 in LNAI. Springer, Berlin, 1996.

[Paulson, 1994] L. C. Paulson. *Isabelle: A Generic Theorem Prover.* LNCS. Springer, Berlin, 1994.

[Pfenning, 1987] F. Pfenning. *Proof Transformations in Higher-Order Logic.* PhD thesis, Carnegie-Mellon University, Pittsburgh Pa., 1987.

[Pitt, 1996] J. Pitt. A WWW Interface to a Theorem Prover for Modal Logic. In N. Merriam, editor, *User Interfaces for Theorem Provers*, pp. 83–90, Department of Computer Science, University of York, UK, 1996.

[Raggett, 1998] HTML 4.0 Specification. W3C Recommendation REC-html40-1998-0424, W3 Consortium, 1998. Available at `http://www.w3.org/TR/PR-xml.html`.

[Redfern, 1998] D. Redfern. *The Maple Handbook: Maple V Release 5.* Springer, Berlin, 1998.

[Richardson *et al.*, 1998] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λ*clam*. In Kirchner and Kirchner [1998].

[Rudnicki, 1992] P. Rudnicki. An Overview of the MIZAR Project. In *Proceedings of the 1992 Workshop on Types and Proofs as Programs*, pp. 311–332, 1992.

[Russell and Norvig, 1995] S J. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 1995.

[Schmidt-Schauß, 1989] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *LNAI*. Springer, Berlin, 1989.

[Searle, 1969] J. R. Searle. *Speech Acts.* Cambridge University Press, 1969.

[Shoham, 1990] Y. Shoham. Agent-Oriented Programming. Technical report, Stanford University, 1990.

[Siegel, 1996] J. Siegel. *Corba: Fundamentals and Programming.* John Wiley & Sons, Inc., 1996.

[Siekmann *et al.*, 1998] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, and V. Sorge. A Distributed Graphical User Interface for the Interactive Proof System ΩMEGA. In R. Backhouse, editor, *User Interfaces for Theorem Provers*, no. 98-08 in Computing Science Reports, pp. 130–138, Department of Mathematics and Computing Science, Eindhoven Technical University, 1998.

[Simon, 1982] H. Simon. *Models of Bounded Rationality.* MIT Press, Cambridge, 1982.

[Slind *et al.*, 1998] K. Slind, M. Gordon, R. Boulton, and A. Bundy. An Interface between CLAM and HOL. In Kirchner and Kirchner [1998], pp. 129–133.

[Smith, 1980] R.G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. In *IEEE Transaction on Computers*, no. 12 in C-29, pp. 1104–1113, 1980.

[Smolka, 1995] G. Smolka. The Oz Programming Model. In J. v. Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pp. 324–343. Springer, Berlin, 1995.

[Steiner, 1992] D. Steiner. MEKKA: Eine Entwicklungsumgebung zur Konstruktion kooperativer Anwendungen. In J. Müller and D. Steiner, editors, *Kooperierende Agenten*, no. D-92-24 in DFKI Document, pp. 17–21. DFKI, Saarbrücken, 1992.

[Steiner, 1997] D. Steiner. An Overview of FIPA'97, 1997. Available at `http://drogo.cselt.stet.it/fipa/fipapres1.zip`.

[Sutcliffe and Suttner, 1997] G. Sutcliffe and C. Suttner. The Results of the CADE-13 ATP System Competition. *Journal of Automated Reasoning*, 18(2):259–264, 1997.

[Thompson, 1991] S. Thompson. *Type Theory and Functional Programming.* International Computer Science Series. Addison Wesley, 1991.

[Weidenbach, 1997] C. Weidenbach. SPASS: Version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.

[Wooldridge and Jennings, 1995] M. Wooldridge and N. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2), 1995.