Coordination of Mathematical Agents
PhD Research Proposal

Jürgen Zimmer

September 25, 2001

Contents

1 Introduction

The aim of the research proposed in this document is the application of the methodology and the techniques of distributed artificial intelligence research to the MathWeb

software bus which has been developed in the AG Siekmann[1]. We intend to develop a distributed network of mathematical software agents (MathWeb agents) which use agent communication languages to exchange information about their (virtual) knowledge bases and to assign subtasks to specific problem solvers. Furthermore, we will investigate the applicability of different agent coordination techniques to societies of MathWeb agents.

1.1 Automated Reasoning

The origin of automated reasoning dates back to the 17th century when Leibniz had the idea to formulate a universal formal language (*lingua characteristica universalis*) in which to express all human thoughts. Leibniz's vision was that with the help of this language and a formal calculus (*calculus rationcinator*) it would be possible to verify human thought like it is possible to verify an arithmetic calculation. This idea cumulated in the famous citation "Calculemus! - Let us compute.".

In 1879, Leibniz' ideas were taken up again in by Frege who defined the first formal language [Fre79]. This language is today known as first order predicate calculus (PL1) and builds the basis of many deduction systems.The first complete calculus for PL1 was presented by Hilbert in 1927 [Hil27]. Hilbert wanted to prove that it was possible to describe and verify all mathematical statements in his logical calculus. Hilbert's calculus was proved to be complete by Gödel in 1930, but it was also Gödel who proved that it is impossible to formalize all of arithmetic in any correct logical system [Göd31]. What remained was the semi-decidability of PL1 which was shown by the work of Herbrand and Skolem. This property of PL1 is the justification for its use in modern deduction systems.

With the development of computer technology and its application to mathematics, it was a consequent step to use computers not only for numeric or symbolic computation, but also to automatically prove theorems with a logical calculus, i.e. to build deduction systems. The first simple deduction systems tried to enumerate the Herbrand universe in order to find a proof for a given theorem, but they where not very successful in proving theorems. One reason for this were the exponentially large search spaces. The development of unification and the resolution principle [Rob65] pushed the problem solving horizon of deduction systems and some systems were even able to prove open problems in combinatorics. Many of todays powerful automated theorem provers are based on resolution and use various specialized heuristics to prune the search space. But they also still fail to prove many theorems of medium complexity, like, e.g. most of the limit theorems [Mel97].

This drawback of classical deduction systems and the insight that humans obviously use special techniques to cope with the huge search spaces led to the fundamental idea of Bundy to first plan proofs at an abstract level before filling in the details of the proof [Bun88]. The research area of *proof planning* was born and since then evolved to a fruitful paradigm shift in automated deduction.

Proof planning also builds the basis of the mathematical assistance system ΩMEGA [BCF+97, SKM99]. The ΩMEGA system is based on *knowledge-based proof planning* [MS99] that is a variant of proof planning as it was introduced by Bundy. Knowledge-based proof planning makes extensive use of mathematical knowledge, which can be encoded in proof planning methods, control rules, or proof strategies. Another important source of mathematical knowledge are external reasoning systems which usually encapsulate knowledge in a special domain of mathematics, e.g. deduction systems

[1] `http://www.ags.uni-sb.de/\~omega`

that are specialized in induction or equational theorem proving, or computer algebra systems containing special computation algorithms. Throughout this document, we will use the term *reasoning system* for both, deduction and computational systems.

The need for the integration of external reasoners into ΩMEGA led to the development of a first simple integration scheme [SHS98] based on the distribution features of the OZ programming language. In recent years, this architecture has advanced to the MATHWEB software bus (MATHWEB-SB) [FK99]. The MATHWEB-SB is now totally independent of ΩMEGA and supports the connection of a wide-range of *mathematical services* by a common software bus. The development of the MATHWEB-SB was not a coincidence. With computer networks becoming ubiquitous in every subfield of computer science, the idea of concurrent, distributed problem solving also became influential in the automated reasoning community. The subfield of distributed automated reasoning (DAR)[2] arose with its own methodology and research projects [Bon00], e.g. PROSPER [DCN+00], TEAMWORK [Den93], and, last but not least, the Logic Broker Architecture [AZ00] project. All these approaches have their individual strengths and limitations. In section 2.1 we give a more detailed description of the architectures and name new desiderata that are not met by any existing system.

1.2 Distributed Artificial Intelligence

Since its early days in the mid and late 1970s distributed artificial intelligence (DAI) has evolved to an established research and application field. While classic AI research is mainly interested in the development of single computer programs that show or emulate some kind of "intelligent" behavior, distributed artificial intelligence is divided into several subfields, the most important is the study, construction and application of multi-agent systems (MAS) [G.W99]. According to RUSSEL and NORVIG [RN95], an *agent* is a self-contained, autonomous computational structure which is sited in a certain physical or virtual environment. Agents are supposed to perceive their environment and act upon it through actions. As an interacting entity, an agent can be affected in its activities by other agents and perhaps humans.

A key pattern of interaction in multi-agent systems is goal- or task-oriented coordination [NLJ96], both in cooperative and in competitive situations. In the case of *cooperation* several agents try to combine their efforts to achieve as a group what the individuals alone cannot achieve. In case of *competition* several agents try to get what only some of them can have.

The long term goals of DAI are to develop and implement mechanisms and methods that enable agents to communicate as well as humans, and to understand the interaction between intelligent entities such as software agents or humans.

Many existing and potential applications of DAI are described in the literature [JSW98]. These range from industrial applications, such as, electronic commerce, and real-time monitoring of communication networks, up to complex research questions, like, for instance, information handling and information retrieval in the Internet, and the investigation of social aspects of intelligence and the simulation of complex social phenomena. These application areas of DAI have in common that they show up one or more of the following characteristics [BG88]:

- Inherent Distribution: The data and information that is processed by the agents

[2] Some authors also use the terms *distributed automated deduction* or *parallel theorem proving*. In this proposal we use the term DAR because we deal also with computation systems (e.g. computer algebra systems).

  - are stored at geographically different locations
  - arise at different times
  - can only be accessed if the agent is familiar with the specific ontologies and languages

- Inherent Complexity: The application domain is too large to be solved by a single, centralized systems because of the limitations of current hardware of software technology.

Enlarging a centralized system for a inherently complex application domain is very difficult, time-consuming, and costly. Such an enlargement usually leads to fragile architectures that break down if the application requirements change only slightly. The alternative way is to distribute the solution process across multiple computational entities (the agents) that are capable of coordination, where the coordination of agents is crucial for the success of the multi-agent system.

1.3 Proposed Research

The research proposed in this document aims at developing a distributed mathematical problem solving system which is robust and scalable and which automatically chooses suitable reasoners for a problem at hand. To reach this goal, we intend to apply the agent-oriented programming paradigm [Sho91] to the MathWeb-SB and build up a world wide web of mathematical software agents, so called MathWeb agents. MathWeb agents are intended to use agent communication languages to exchange information about their (virtual) knowledge bases and to assign subtasks to specific problem solvers. Having the means of agent communication, we are going to develop a general communication framework for MathWeb agents which takes into account a formal specification of mathematical services, the context of agent conversations, and the role of mathematical knowledge bases in these conversations. Furthermore, we are going to investigate the applicability of different agent coordination techniques to societies of MathWeb agents with regard to distributed, decentralized, and autonomous solving of mathematical problems.

The proposed research will combine the ideas, methodologies, and techniques of two more or less independent subfields of AI, automated deduction and distributed artificial intelligence. The research will be supervised by Prof. Dr. Jörg Siekmann.

1.4 General Framework

The "Arbeitsgruppe Siekmann" (AGS) of Prof. Siekmann offers a excellent scientific environment for the proposed research. It has many years of experience in automated and semi-automated theorem proving and in proof planning. Developing the ΩMEGA [BCF+97] system the AGS could already gain much experience with knowledge-based proof planning, multi-strategy proof planning, and the integration of various external reasoning systems. With the MathWeb-SB the AGS developed a robust platform for this integration and for the inter-operation of a wide range of mathematical services in general. Therefore, the group has already gained significant experience with the inter-operation of heterogeneous reasoning systems. The MathWeb-SB also builds the basis of the DORIS project [BBK99] of the computational linguistics department of the *Universität des Saarlandes* (USAAR) and in the *ActiveMath* project [Mel00] of the DFKI.

Also the OMDoc format for open mathematical documents has been developed by members of the AGS. It is an extension of the OpenMath standard and is especially suited for communicating mathematical objects, such as axioms, definitions, theorems, and whole theories, between mathematical software systems and mathematical agents.

First work towards agent-based theorem proving and the combination of interactive and automated theorem proving has been done with the Ω-ANTS command suggestion mechanism [BS98] and the Ω-ANTS theorem prover [BS00]. The Ω-ANTS approach also make extensive use of external reasoners integrated via the MathWeb-SB.

The AGS is a member of the international research network Calculemus which brings together researchers from the fields of computer algebra systems and automated theorem provers. The goal of Calculemus is the development of a new generation of mathematical assistance systems based on the integration of the deduction power of deduction systems and the computational power of computer algebra systems. The AGS and several other research groups of the Calculemus network are supported by an IHP network grant of the European Union which allows the exchange of researchers and cooperations, for instance, with the research groups of Prof. Bundy in Edinburgh, Prof. Buchberger in Linz, Prof. Giunchiglia in Trento, and with the Mechanized Reasoning Group (MRG) in Genoa.

In the surroundings of the USAAR, there is also the Multi-Agent Systems Group of the German Institute for Artifical Intelligence (DFKI) at the University of the Saarland which has an internationally respected expertise in the theory and practice of multi-agent systems and the application of multi-agent programming to different domains.

1.5 Structure of this Document

This proposal is organized as follows. In section 2 we first describe requirements of modern applications of automated reasoning in general and give an introduction to current research projects in the field of distributed automated reasoning. Section 2.2 is dedicated to the MathWeb-SB because it builds the basis for this research. We then name some desiderata that are not met by any of the currently available architectures for distributed automated reasoning. In section 3 we shortly depict the ideas of DAI and focus on the agent communication language KQML and on coordination techniques for multi-agent systems. Section 4 describes the concrete goals of this research. We close this document with a short summary and a work plan in section 5.

2 Automated Reasoning

Modern applications of automated reasoning and theorem proving, for instance in the mathematical assistance system ΩMega or in program verification [HLS+96], call for open, distributed architectures which allow the integration of specialized reasoning systems. A widely accepted approach to build such architectures is to upgrade classical reasoning or computation systems to so-called *mathematical services* [HC96] by providing it with an interface to a common *mathematical software bus* [CH97]. Many research projects in the field of distributed automated reasoning followed this approach and developed different architectures for distributed automated reasoning. Franke et al. [FHJ+99] name four major requirements for distributed systems of mathematical services:

Modularization: Deduction systems are very complex and specialized AI programs which are typically developed by more than one individual. Usually the developers of deduction systems are less interested in developing a modularized system

with standardized interfaces but in constructing an efficient standalone reasoning system which can be used as a black-box. For building a distributed system of reasoning components, it is important that the components offer a standardized interface which encapsulates related functionality into re-usable modules.

Inter-Operability: Inter-operability is *the* central presumption for the construction of a working distributed system out of heterogeneous components. Inter-operability depends on a common platform which supports the exchange of mathematical services, such as, for instance, the CORBA middle-ware [Sie96]. In a system of inter-operable services, each service provides additional functionality to the system as a whole and, in turn, can use all existing services to use for its own reasoning.

Robustness: Software systems with a fixed inflexible architecture often have problems with handling failures. A classical proof system with a static topology will not work if one of its parts does not work. To build a robust system a dynamic, decentralized architecture is needed which can permanently provide mathematical services, even if some parts, e.g. a particular service, are temporarily shut down.

Scalability: Computer networks, whether they are local (LAN) or global (WAN, Internet) typically show a dynamic allocation of computational resources. Thus, for distributed systems to perform in an optimal manner, it is important that the system adapts its topology and the distribution of tasks to the number of tasks to be solved and to the currently available resources, i.e. to find an optimal load-balancing. Consequently, also a distributed reasoning architecture should show a flexible, dynamic topology which adapts to changing computational resources.

The central claim in [FHJ+99] was that the Agent-Oriented Programming paradigm meets all these requirements. The authors of [FHJ+99] therefore propose to apply this paradigm to the MATHWEB software bus. They mainly focus on the communication between logical reasoning systems using the agent communication KQML. In section 2 we will argue that sole communication with an agent communication language is not sufficient to build a multi-agent system, in which *autonomous* agents act in a *coherent* way and perform *decentralized* mathematical problem solving. But first, we present a short survey of research projects in the field of distributed automated reasoning in the following sections. We briefly describe the goals and the results of the projects and which of the above-mentioned requirements they meet. In section 2.2 we give a more detailed description of the MATHWEB software bus, because it builds the basis for the ideas presented in section 4.

2.1 Architectures for Distributed Automated Reasoning Systems

The PROSPER project [DCN+00] aims at developing the technology needed to deliver the benefits of formal specification and verification to system designers in industry. The central idea of the PROSPER project is that of a *proof engine* (a custom built verification engine) which can be accessed by applications via an application programming interface (API). This API is supposed to support the construction of CASE-tools[3] incorporating user-friendly access to formal techniques. Much work of the PROSPER project went into the definition and implementation of the API, the development of the PROSPER-toolkit, and into intensive case studies. The PROSPER-toolkit currently integrates the

[3]CASE: Computer Aided Software Engineering

higher order theorem prover HOL, and the verification system ACL2. With respect to the requirements mentioned in the last section, the API of PROSPER mainly allows the modularization of CASE software systems and the inter-operation of these modules with proof engines. However, the components of PROSPER toolkit still use a proprietary protocol for communication.

The *Logic Broker Architecture* (LBA) [AZ00] has been developed within the MRG[4]. It is based on the communication functionality provided by the *Common Object Request Broker Architecture* (CORBA) [Sie96, Groa] and the OPENMATH standard [CC98]. CORBA is a widely used standard for the development robust, platform and language independent client-server applications. The LBA provides the infrastructure needed for making mechanized reasoning systems inter-operate by a simple registration/subscription mechanism. Additionally, it offers location transparency and a translation mechanism which ensures the transparent and provably sound exchange of logical services. Logical services in the LBA (e.g. `factorize` for the factorization of polynomials) are abstract, i.e. they are independent of a concrete implementation in a reasoning system (e.g. a concrete algorithm which performs the polynomial factorization). It is planned to insure the logical correctness of the interaction of services by using a *logic service matcher* which tries to find a morphism between the logic of a client reasoning system and the logic of a service server (see [AZ00] for details).

The TEAMWORK method [Den93, AD93] has been developed to distribute equational theorem proving by completion. A *team* in TEAMWORK consists of several *experts* which are the actual problem solvers (e.g. equational theorem provers with different heuristics) and of *referees* who evaluate the achievements of the different experts and determine their best results. A central *supervisor* composes the group of agents working on a given problem and composes the initial search state for these agents. One of the main motivations for the TEAMWORK method was the fact, that communication overhead can easily diminish the merits of distributed problem solving. In the TEAMWORK method the amount communication is reduced by introducing phases of the overall problem solving process, so-called *team meetings*, in which the reasoning agents send their best results to the central supervisor. Between two team meetings, in the *working phases*, the agents compute new local results (sets of clauses) and no communication takes place at all. The TEAMWORK method led to synergetic effects which allowed the system to perform better that every single theorem prover alone. However, TEAMWORK was based on a network of homogeneous theorem provers which differed only in their search heuristic.

The TECHS system [DD98] builds on TEAMWORK technology and combines *heterogeneous* state-of-the-art theorem provers while minimizing the changes that have to be done to these provers. TECHS combines the ATPs SPASS and DISCOUNT and the tableau-based prover SETHEO that communicate by exchanging clauses. The provers perform two kinds of cooperation. They send requests for needed information (*demand-driven*) and autonomously send information they found useful to all other agents (*success-driven*). In order to reduce the communication overhead several heuristics are used to select the clauses that sent to other agents and clauses from other agents that seem to be useful. TECHS has also been combined with the ILF environment to allow interactive cooperation of a human user with the prover network.

The Ω-ANTS theorem proving approach [BS00] is based on the homonymous command suggestion mechanism within the ΩMEGA system. The core of Ω-ANTS suggestion mechanism is a central hierarchy of a suggestion-blackboard and several command-blackboards. Command argument agents write information about a central proof data

[4]MRG: Mechanized Reasoning Group, DIST, Università di Genova, Italy.

structure on the latter. Other argument agents are triggered by this information and in turn write their suggestions to the command-blackboard. The suggestion agents read from the argument-blackboards and finally write command suggestions on the suggestion-blackboard. A human user can then select one of the commands suggested in the current proof state. BENZMÜLLER and SORGE automated this process to a full automated theorem proving procedure by adding an agent which automatically selects a command and stores all other selections for possible backtracking. With special agents, external reasoning systems are also integrated into the proof search. For this, Ω-ANTS relies on the facilities of MATHWEB-SB. Ω-ANTS provides an any-time algorithm for command suggestion and has proved to be quite flexible. Agents can be defined in a declarative way and can be added, deleted and modified during run-time.

Summary. The TEAMWORK method, the TECHS system, and the Ω-ANTS approach are closest to the research proposed in this document. But these approaches have some drawbacks. While the TEAMWORK method and the TECHS system are still restricted to resolution-based theorem provers, the Ω-ANTS theorem prover relies on a central proof object and on communication via blackboards and therefore suffers from the typical problems of centralized architectures like restricted parallelism and bottlenecks. Furthermore, the agents in both approaches do not communicate via a standardized communication language but via some proprietary protocol.

This research goes one step further in that it aims at the development of fully autonomous, heterogeneous mathematical reasoning agents that communicate via standardized languages and encapsulate both, deduction systems and computation systems. We will also apply higher order coordination protocols (like the contract net, cf. section 3) to our societies of agents which has not been done in previous approaches. Our reasoning agents will be capable to perform distributed mathematical problem solving with decentralized control and a decentralized construction of a proof object.

2.2 The MATHWEB Software Bus

Also the MATHWEB Software Bus (MATHWEB-SB or short MATHWEB) is a platform for distributed automated theorem proving that supports the connection of a wide range of *mathematical services* by a common software bus [FK99]. The MATHWEB-SB provides the functionality to turn existing theorem proving systems, computer algebra systems, and tools into mathematical services that are homogeneously integrated into a networked proof development environment. The environment thus gains the services from these particular modules, but each module in turn gains from using the features of other, plugged-in components. The MATHWEB-SB is implemented in the concurrent constraint programming language MOZART [Smo95, grob].

The development of the MATHWEB-SB originates in the effort to integrate external reasoning systems into the mathematical proof assistance system ΩMEGA. The first version of MATHWEB-SB mainly enabled the user of ΩMEGA to run several ATPs in parallel on a problem in order to maximize the likelihood of success and to minimize the time the user has to wait for a response.

In the last three years we further developed the MATHWEB system which is based on the MATHWEB-SB technology and created a stable network of mathematical services which is in every day use. The services of the current MATHWEB system are used permanently by many projects, e.g. the ΩMEGA project, the DORIS [BBK99] system, and the *ActiveMath* project [Mel00]. MATHWEB currently integrates many different reasoning and computation systems:

Automated Theorem Provers (ATP). MathWeb currently features the first-order theorem provers *Bliksem*, EQP, Otter, ProTeIn, Spass, Waldmeister and the higher-order theorem prover TPS (see [ABI+96] for reference). Furthermore, there is a service `competitive-atp` that calls concurrently a given set of ATPs on a set of first order problems. This service uses the round robin method to distribute unsolved problems to ATPs that are not busy.

Computer Algebra Systems (CAS). There are services wrapping the CASs Maple, MagMa, CoCoA, and GAP (see [KKS98]). These CASs are successfully used to perform computations for the proof planner of the ΩMEGA system. Their computational power is essential for planning proofs of limit theorems [Mel97, Zim00] and for theorems on properties of residue classes [MS00].

Mediators. Mediators are mathematical services that transform mathematical knowledge from one format to another. MathWeb integrates translation services which translate mathematical formulas from one language to another. However, the current translation services translate formulas from one proprietary language to another (e.g. from POST, the logic underlying the ΩMEGA system, to Maple syntax) and do not use standardized content languages like OpenMath [CC98] or OMDoc [Koh, Koh00].

Mathematical Knowledge Base. MathWeb currently 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. MBase is still under development but a preliminary version already serves mathematical documents to the *ActiveMath* system.

Constraint Solvers. MathWeb currently offers two constraint solving systems. $\mathcal{CoSIE}$ [Zim00, MMZ00] is a constraint solver for non-linear arithmetic constraints over the real numbers. Chorus [KN00] is a special constraint solver developed in computational linguistics which handles dominance constraints to resolve ambiguities in natural language sentences.

The current structure of the MathWeb system is depicted in Figure 1. The software

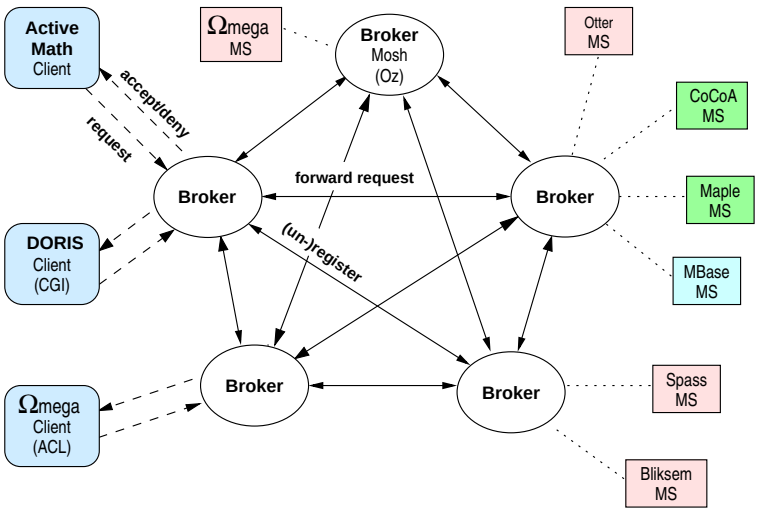

Figure 1: Current state of the MathWeb system

bus functionality of MathWeb is realized by a model quite similar to Corba[5] in

[5]However, it is important to note, that the current implementation of MathWeb is *not* not based

which local *brokers* provide routing and authentication information to the mathematical services (see [FK99, SHS98] for details). So called *meta-services* (MS), offer the mathematical services (e.g. an ATP, CAS or a model checker) to their local broker. MATHWEB brokers register and unregister each other and, therfore, build a dynamic web of brokers. A MATHWEB broker forwards service requests, if the requested service is not offered locally by one of the broker's meta-services.

Client applications, like, for instance, the ΩMEGA system, ΩUI (a GUI for ΩMEGA), or a CGI-script, connect to one of the MATHWEB brokers and request services. If the requested service is not offered by a local meta-service, the broker forwards the request to all other brokers until the service is found (`accept`) or it isn't found anywhere in the MATHWEB (`deny`). If the requested service is found, the client application receives a reference to a newly created *service object* and can directly send messages to the object. Service objects, like e.g. the ΩMEGA system, can again act as MATHWEB clients and request other services.

2.3 Additional Desiderata

The MATHWEB-SB already meets some of the desiderata mentioned in section 2. It allows modularization and inter-operability of mathematical services. The dynamic (un-)registering of MATHWEB brokers and meta-services allows to build a robust system with a flexible topology. However, the current version of the MATHWEB-SB does not allow to build a scalable system which adapts its topology to changing computational resources, i.e. MATHWEB-SB is not resource adaptive. In the following sections, we propose some additional desiderata that are also not met by the current version of the MATHWEB-SB. These desiderate originate in the experience we made in recent years with the integration of different reasoning and computation systems.

2.3.1 Abstract Mathematical Services

A mathematical service in MATHWEB mainly consists of the reasoning system itself which is encapsulated into a MOZART wrapper that handles the communication. To access the reasoning system, a MATHWEB client application has to send the right commands to the system, i.e. the client has to know the system's internal structure, commands and input syntax.

Example 2.1: The CAS MAPLE is integrated as a mathematical service into MATHWEB and is used to solve computational problems in proof planning such as simplification of terms and polynomial devision. To perform these computations, the specific commands of MAPLE have to be called (`simplify` or `quo` respectively) in the MAPLE specific input syntax. Naturally, the commands and the input syntax differ from CAS to CAS. For instance, in the Computeralgebra-System CoCoA the command for polynomial devision is called `DivAlg` and the input syntax is different from the syntax of MAPLE.

Abstract mathematical services would free the designer of reasoning systems from the burden to learn the internals of all integrated reasoners. Abstract services should be independent of concrete implementations (e.g. of polynomial devision in MAPLE and CoCoA) and should provide a system independent interface with a standard I/O language, e.g. the OPENMATH standard. First experiments with abstract mathematical services have already been done in the context of the LBA (c.f. section 2.1).

on CORBA middle-ware, since their is no CORBA implementation for MOZART available at the moment.

2.3.2 Autonomy and Decentralization

Classical integration of reasoning systems (e.g. the integration of ATPs and CASs in ΩMEGA) follow a master-slave model, i.e. one master system sends subtasks to the slave systems which act more or less as a black-box. The master system has full control over the slaves which cannot communicate with each other. Very often, they even don't know of the existence of other reasoning systems. We say, the reasoning process is *centralized*. Mostly, the master-slave communication is also synchronous, i.e. the master waits for the results of its slaves and is blocked until the results of the slave systems arrive.

The following example will show that a decentralization of problem solving process and an asynchronous communication between autonomous systems can help to reduce communication, can lead to more parallelism in problem solving, and can release the reasoning systems from waiting for the results of external reasoners.

Example 2.2: The higher-order theorem prover TPS [AINP90] is currently integrated in a master-slave manner into ΩMEGA. TPS has a flexible mechanism for the expansion of definitions, i.e. during the search for a proof, definitions for symbols can be expanded *by-need*. When ΩMEGA wants an open subgoal to be solved by TPS it first retrieves the definitions of all symbols in the subgoal from its local knowledge base and sends these definitions to TPS, even if they are not (all) needed for the proof attempt of TPS. Finally, ΩMEGA sends the actual subgoal and waits for TPS' result. It turned out that a great portion of the total problem solving time is needed for sending all definitions to TPS. The amount of communication could be drastically reduced, if the symbol definitions could be delivered by a separated mathematical knowledge base MBASE. Then an *autonomous* version of TPS could request the definition of a symbol only if it is actually needed. This communication between TPS and MBASE would not affect the ΩMEGA system anymore. If, additionally, the communication between ΩMEGA and TPS was asynchronous, ΩMEGA would only have to send the actual subgoal to TPS and go on trying to solve other subgoals until the result of TPS arrives.

MATHWEB agents should therefore exhibit some autonomy, i.e. they should be able to act on their own, without the intervention of other systems. For instance, in Example 2.2 TPS should be able to dynamically request the needed definitions from a mathematical knowledge base. Reasoners should also be able to decide whether they can carry out a given reasoning task and whether they accept a task or not. A reasoning system could, for instance, deny a task if it does not have enough computational resources left to spend.

2.3.3 Coordination

In all architectures mentioned in section 2.1 and also in the MATHWEB-SB, the designer of a reasoning systems has to coordinate the use of external reasoning systems, i.e. he has to decide which external reasoning systems to call on which subproblem, and which of the various functionalities of the external reasoner to use (e.g. the CAS MAPLE 6 offers more than 3000 computational functions). In most cases, it is not obvious whether a reasoning system can perform a given task or not, or which reasoner will perform best on this task.

With an accurate formal specification of reasoning tasks and the capabilities of reasoning systems it is possible to – at least partially – automate the coordination of different reasoners. This automation could lead to a system of reasoners which is much more flexible and whose components would dynamically coordinate their behavior for a

given problem at hand and not according to a previously fixed integration scheme. The dynamic coordination of MathWeb agents could also take into account the previous performance of an agent on certain classes of problems in order to choose the best agent for a given task. AI learning techniques could be used to learn an optimal assignment of tasks to MathWeb agents.

The problems of resource-adaptivity, autonomy, decentralization, and coordination are central research topics in the field of distributed artificial intelligence (DAI). In the following section we give a short overview over DAI research. In section 4 we describe how we intend to apply DAI techniques to distributed automated reasoning and name the central research questions that follow from this application.

3 Distributed Artificial Intelligence

While the classic AI endeavor is mainly interested in the development of single computer programs which show or emulate some kind of "intelligent" behavior, distributed artificial intelligence (DAI) is concerned with the study, design and application of distributed problem solving [G.W99]. There have been many proposal and, at least, as many discussions on what an *agent* actually is. Following Russel and Norvig[RN95], an *agent* is a self-contained, autonomous computational structure situated in a physical or virtual environment. An agent can perceive it's environment through *sensors* and act on it with *effectors*. Wooldridge and Jennings [WJ95] proposed the following key properties for the characterization of agents:

Autonomy: Agents should to some extent have control over their behavior and should act without the intervention of humans or other software systems.

Reactivity: Agents should react to some changes in their environment. In the case of software agents that live in a virtual environment this means that the agents should be able to modify their behavior according to changing environmental and computational constraints (e.g. resources, like time and memory).

Pro-activeness: Agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives.

Social Ability: Agents are capable of interacting with other agents (and possibly humans) to satisfy their design objectives and reach their goals. This requires that agents have the means to communicate with other agents and that they have some kind of *social model*, i.e. knowledge about neighboring agents and their capabilities.

In the following sections, we solely talk about *software agents* which live in a (virtual) software environment. The development of software agents was strongly influenced by the paradigm of Agent-Oriented Programming.

3.1 Agent-Oriented Programming

The term *Agent-Oriented Programming* (AOP) was coined by Shoham in 1991 [Sho91]. He described it as a "*new programming paradigm, based on a societal view of computation*". The key idea is to directly program software agents which encapsulate arbitrary, traditional software applications. These *agent-shells* are able to interface and control the operation of the embedded services. The agents introduce a social model referring to other service agents with which they build a *society of agents*. The basic means

for the interaction between social agents is a common *Agent Communication Language* (ACL) which enable the agents to coordinate their behavior, i.e. steer the embedded applications by exchanging *beliefs*, *goals*, and *intentions*. There have been many proposals for agent communication languages. Two of the most widely used are the FIPA industrial standard [Ste97, fIPA] for physical agents and the KQML standard for software agents. These two ACLs are equally expressive but differ in syntax and application domains. While FIPA is supposed to be applied in industrial domains, such as telecommunication, KQML is widely used in research projects. Since we intend to build software agents, we think, that KQML is the ACL of choice for our application domain. Last but not least, KQML and FIPA have a similar expressiveness and offer a similar set of performatives. Since the functionality that has to be implemented for KQML is very similar to that of FIPA, we think that it will not be much effort to offer both communication languages in the future.

3.2 The Knowledge Query and Manipulation Language

The Knowledge Query and Manipulation Language (KQML) [FMF92, Lab96] is a communication language for software agents which supports the exchange of information about the (virtual) knowledge bases of the agents. KQML is both a message format and a message-handling protocol to support shared knowledge in a multi-agent system. KQML is based on the *speech act theory* [Sea69]. Its primitives are so called *performatives* which define the permissible "speech acts" that actions are allowed to perform in communication with each other. Thus, KQML messages do not solely communicate sentences in some language, but rather communicate an attitude about the content of the message.

KQML performatives can be modeled as actions which change the cognitive states of agents. According to [Lab96], cognitive states can be specified using the predicates *know, want, intend,* and *bel* which describe the knowledge, goals, intentions, and beliefs of agents. With these predicates, the semantics of KQML performatives can be formally specified in terms of preconditions and postconditions describing the applicability conditions and the effect of the performatives respectively (cf. [Lab96]).

A society of KQML speaking agents can be enriched with special agents, called *facilitators*. Facilitators typically provide functionalities such as: association of physical addresses with symbolic names of agents, registration of agents, and forwarding and brokering of messages.

3.3 Coordination in Multi-Agent Systems

In most multi-agent systems sole communication via an ACL is not sufficient to ensure that the agent community acts in a *coherent* way, where *coherence* refers to how well a system of agents behaves as a unit [Syc89]. A serious problem in MAS is to maintain global coherence without explicit global control. In this case, the agents must be able on their own to determine goals they share with other agents, determine common tasks, avoid unnecessary conflicts, and pool knowledge and evidence. *Coordination* can take place in societies of antagonistic agents (*negotiation*) and of non-antagonistic agents (*cooperation*) as shown in Figure 2. Typically, to cooperate successfully, each agent must maintain a list with the capabilities of the other agents, and also develop a model of future interactions. This presupposes sociability of agents [HL99].

One means for achieving coherence are higher-level interaction protocols. These interaction protocols govern the exchange of a series of messages among agents – a *conversation* – and help the agents to coordinate their behavior. Several coordination

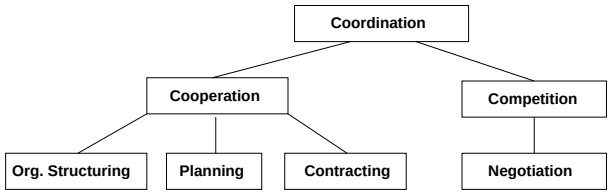

Figure 2: Different kinds of coordination

techniques and protocols have been devised for multi-agent systems so far. NWANA, LEE, and JENNINGS give an overview in [NLJ96]. For a system of competitive agents several forms of negotiation have been proposed. They are mostly based on *game theory* [LR57], local planning [KvM91], or on theories of human negotiation [BM92].

Organizational Structuring. This is one of the simplest coordination scenarios which exploits an a priori organizational structure. The organization implicitly defines the agent's responsibilities, capabilities, connectivity and control flow. One of the most widely used forms of organizational structuring is the *blackboard coordination* where agents post to and read from a central blackboard [HR85]. Using a blackboard the agents' behavior is triggered by information written on the blackboard by other agents. Blackboard coordination, in its extremes, mitigates against all the benefits of DAI: parallelism, reliability, robustness, minimal bottlenecks, etc. Additionally, as stated in [NLJ96], all agents in a blackboard architecture "[...]*must have a common domain understanding (i.e. semantics). For this latter reason, most blackboard systems tend to have homogeneous and rather small-grained agents.*".

Multi-agent Planning. In multi-agent planning, the agents of a society locally plan their future actions and try to coordinate their behavior by combining the local plans to a consistent global plan. In *centralized multi-agent planning* [CMS83] the agents first compute local plans and send them to a central *coordinating agent*. On receipt of all (partial) local plans, the coordinating agent analyzes them in order to identify potential inconsistencies and conflicting interactions. The coordinating agent then tries to modify the partial plans and to combine them into one multi-agent plan without conflicting interactions. In *distributed multi-agent planning* [CL81] the resolution of conflicts is performed by all planning agents who communicate during planning to build and update their individual plans. Both variants of multi-agent planning have two major drawbacks: 1) The agents must share and process huge amounts of information, i.e. the planning process is very communication intensive. 2) In many domains, the combination of local plans to one consistent global plan is a very complex and time-consuming task.

Contracting. A now-classic and intensively studied coordination technique is the *Contract Net Protocol* (CNP) introduced by SMITH [Smi80, DS83]. The CNP is based on a decentralized market structure. Agents can assume two roles: They can act as *managers* who break problems into subproblems and search for *contractors* to perform the subtasks. Or they act as contractors who perform tasks. Contractors may recursively become managers and further decompose their subtask and contract the tasks resulting from this decomposition to other agents. The core of the CNP is a task announcement and bidding process which consists of five steps:

1. A manager announces a task
2. Contractors evaluate the task w.r.t. their abilities and commitments
3. Contractors send bids to the manager
4. The manager evaluates the bids and chooses one or several contractors and awards the contract for the announced task to them.
5. The contractors send the result of the task performance to the manager

In most applications of the CNP the manager announces an abstract specification of a task in the first step. This specification is highly domain-dependent and contains all information relevant for the contractors to determine their bid for the task.

The CNP has been used in many applications, e.g. [Par87]. It turned out that it is best suited for domains where

- the application task has a well-defined hierarchical nature;
- the tasks have a coarse-grained decomposition;
- there is minimal coupling between subtasks.

Since there is no formal model for task announcing, bidding and awarding in the original work, the CNP was also further developed and extended by SANDHOLM [San93] and others. HUHNS and SINGH have show that the CNP is a high-level coordination strategy which also provides the means for the distribution of tasks and for self-organization within a group of agents [HS94].

4 Agent Technology for Distributed Mathematical Reasoning

The long-term goal behind this proposal is to apply the techniques of Distributed AI (e.g. AOP and coordination) to distributed automated reasoning in order to develop a network of heterogeneous mathematical agents. These MATHWEB agents shall have a social model of their environment and communicate with an agent communication language and several more or less standardized or specialized content languages. The MATHWEB agents should perform autonomous distributed problem solving with a decentralized control, handle shared proof objects, and dynamically coordinate their behavior given a problem at hand. The problem solving process should be resource-adaptive in an optimal manner, i.e. it should take into account all computational resources (e.g. CPU-time, free memory, network bandwidth) that are available to the different MATHWEB agents. A net of MATHWEB agents would offer the means for new research projects, such as distributed proof planning. Last but not least, agent-oriented programming has shown to be a perfect paradigm for human computer interaction (HCI) [BW94]: Since agent communication languages like KQML are based on models of human communication a human user can simply be modeled as another agent in the multi-agent system.

This ultimate goal is very ambitious and beyond the scope of a single PhD thesis. In the following, we therefore restrict the aim of the proposed research to a list of feasible goals that are basic for the development of working MATHWEB agent societies. The idea is to apply the AOP paradigm to the MATHWEB-SB. Following [FHJ+99], we will to encapsulate MATHWEB's mathematical reasoning systems into an agent shell

in order to build mathematical reasoning agents, the *MathWeb agents*. With reasoning systems extended to MathWeb agents the interaction between these systems can be modeled as a KQML *conversation* between corresponding MathWeb agents. We think that proposed extensions of the MathWeb-SB can meet the desiderata described in section 2.3. Having MathWeb agents, we will investigate the applicability and the effectiveness of several coordination techniques to decentralized automated reasoning in a society of heterogeneous deduction and computation agents. We will apply multi-agent planning to the multi-strategy proof planner MULTI to build a distributed multi-agent proof planning environment. We will also investigate how the mathematical knowledge base MBase can support abstraction in agent communication in general and in multi-agent planning in special.

The proposed research is a first significant step towards our long-term goal and we shall see how far we can go in one piece of work. In the following sections we describe in more detail the central research problems that have to be solved to develop an efficient society of MathWeb agents and we present some first ideas for a solution of these problems. Of course, the ideas presented here sketch only a starting point for the proposed research.

4.1 MathWeb Agents

The central entities in our research will be MathWeb agents which are reasoning specialists with certain properties. According to section 3 MathWeb agents should have at least the key characteristics:

Autonomy: MathWeb agents should act without the intervention of humans or other agents. On the other hand, agents should not simply act like slaves in the master-slave integration model but should decide for themselves if they are capable and willing to work on a reasoning problem that is sent to them (e.g. an open conjecture to prove).

Social Ability: MathWeb agents should be capable of communicating their goals and needs to other MathWeb agents interacting with other agents by an agent communication language. Additionally, MathWeb agents should be able to understand multiple content languages. They should also have a social model of the reasoning capabilities of other agents (e.g., induction proving specialists, special CAS algorithms).

Pro-activeness: Next to all their social activity, MathWeb agents should, of course, not forget to act goal oriented, e.g. in the attempt to prove a theorem or to perform a computation.

Reactivity: Since MathWeb agents are software agents, a change in the environment means, for instance, changing computational resources like available memory and CPU-time. Agents should react to some changes in their environment. A paragraph in section 4.2.1 is dedicated to resource adaptivity.

Figure 3 shows a first proposal for the structure of MathWeb agents. This preliminary structure addresses the first two of the characteristics mentioned above. Of course, the final version of MathWeb agents should address all four characteristics. Essentially, each agent should consist of the classical reasoning or computation service which is extended by an agent shell. The agent shell is responsible for all communication issues of the agent, i.e. for composing and sending correct KQML messages

and translating incoming messages into concrete actions for the reasoning system. We suppose that each MathWeb agent has its own little database which contains specifications of theoretically existing mathematical services which allows the agent to build reasonable messages.[6] The *conversation module* of an agent handles all ongoing conversations the agent is involved in. A central control unit accesses the database of service specs to create the actual KQML messages, asks the conversation module to start new conversations, and handles incoming messages.

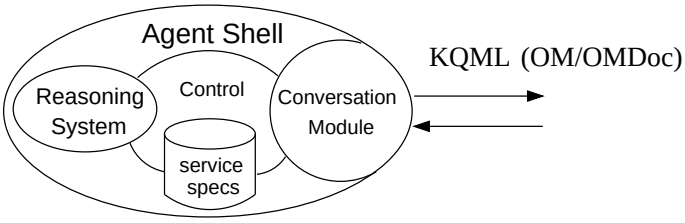

Figure 3: Structure of a MathWeb agent

MathWeb agents are supposed to communicate via KQML with OpenMath, OMDoc, or KQML itself as a content language. OpenMath is a standard for the purely semantic representation of mathematical formulas. Mainly, it is a restricted first-order language and offers variables, constants, applications, bindings, etc. OMDoc aims at the representation of full mathematical documents, including, e.g. plain text, definitions, theorems, and proofs. Both languages commit to a global ontology because every symbol belongs to a OpenMath *content dictionary* (CD) which provides a formal or informal description of the semantics of the symbol. Thus, OpenMath and OMDoc are based on a fixed ontology accepted by most members of the automated reasoning and of the computer algebra community. Therfore, we think that OpenMath and OMDoc are more suitable as content languages for the communication between MathWeb agents than, for instance, the very general *Knowledge Interchange Format* (KIF) [Gea92], which has been used in many other DAI applications, or the MathML standard [IM98], which is mainly concerned with LaTeX-like presentation mark-up of mathematical formulas. But, however, in some applications it might be necessary to develop new content languages in the future. Since the standard language for the encoding of OpenMath and OMDoc is the the *eXtensible Markup Language* (XML [BPSM97]), it is a consequent step to chose Xml also for the encoding of KQML messages to allow a uniform handling of both, the messages and their content. However, Xml encoding of information tends to be about 10 times bigger than an equivalent binary encoding. Therefore, it might be reasonable to additionally allow more efficient content languages for groups of reasoning agents that can communicate with these languages. For instance, a local society of first-order ATPs that all rely on PL1 with equality could easily communicate with an efficient encoding of first-order clauses as a message content.

The proposed MathWeb reasoning agents do not store any knowledge about the capabilities of other agents. We suppose that this knowledge is managed by the KQML facilitators. However, MathWeb agents store knowledge about mathematical services available in principle but they do not know in advance which agents offer these services.

[6] In the future, the service specifications could also be stored in a central MBase which is accessible to all agents.

4.2 Communication between MathWeb agents

We first focus on the problems that are related to the communication between MATHWEB agents. In the following we treat technical problems, the problem of mathematical service specification, and the handling of context in agent communication.

4.2.1 Technical Issues

Communication Overhead. As already mentioned in section 2.1, one of the main problems in distributed equational theorem proving is the communication overhead. Whenever distributed problem solving agents spend most of their computational resources into communication, the benefits of parallelism can be drastically diminished.

The TEAMWORK approach (cf. section 2.1) proved to be successful in the case of distributed equational theorem proving. In this special domains, homogeneous ATPs can work independently on a problem and deliver uniform results (derived clauses) which can be combined w.r.t. some heuristics.

Since we are interested in the integration of heterogeneous systems (e.g. ATPs, CASs, model checkers, or proof planners) the TEAMWORK approach is not appropriate. So, other mechanisms must be used which enable synergetic distributed problem solving and reduce communication to a minimum.

Resource Adaptivity. Adaptivity to changing computational resources is crucial if one wants to build a scalable system of distributed reasoning systems and to ensure the optimal performance of such a system. This is due to the fact that the performance of a reasoning system is *not* merely dependent on powerful algorithms or sophisticated search techniques but also on the available resources. However, an optimal coordination of a society of MATHWEB agents according to the available resources is a very complex – if not infeasible – task. As a first approach to tackle this task we are going to extend MATHWEB agents with a resource module which permanently keeps track of the currently available CPU time and free memory on the local machine. Combined with coordination techniques such as the CNP MATHWEB agents can then use resource information to decide whether they accept a given task or not. This allows for a decentralized and flexible resource handling with a minimum of additional communication.

4.2.2 Characterization of Reasoning Capabilities

As stated in section 2.3.1, it is necessary to define abstract mathematical services in order to free the designer of reasoning systems from the task of learning the internal structure of other reasoners. Up to now, no standard specification language for mathematical services has been developed. For our first experiments with MATHWEB agents and KQML communication, we used special OPENMATH symbols for the specification. This choice was done in analogy to already existing symbols in OPENMATH content dictionaries, e.g. the symbol `factor` in the CD `polyd` that stands for the factorization of polynomials. One advantage of using OPENMATH symbols is the commonly accepted (informal) semantics that is defined in the publicly available content dictionaries.

At the moment, it is not clear, whether the approach of using OPENMATH symbols is appropriate for all mathematical services developed in the future or if we have to develop a new specification language. One problem with OPENMATH symbols is the informality of their semantics. Usually, the developers of content dictionaries only describe the semantics of symbols with at most a handful of sentences in natural language. There is strong evidence that in the future we need a higher-order specification

of reasoning services. However, a specification language for services could, for instance, use OpenMath symbols as a basis to build more complex service descriptions.

Another open problem is the storage and retrieval of the knowledge about mathematical services. As a first answer, we suggest that this knowledge should be located at the KQML facilitators. One could also think of storing this knowledge in a mathematical knowledge base like MBase.

4.2.3 Context in Mathematical Communication

Very often conversations between reasoning systems do not only consist of a single query with a single answer. As we have seen in Example 2.2, in some cases a *context* of a conversation has to be built up because some facts (e.g. definitions, lemmas, or proof assumptions) have to be known by the conversation partner. Also in the communication with Computer Algebra Systems, a context can be built. The CAS Maple, for instance, offers the `assume` facility to define certain preconditions for its computations [7]. However, building a general model for the contexts of MathWeb agent conversations is not easy. Especially, the following questions must be answered:

- Which mathematical knowledge is typically stored in a context?
- Where is this knowledge stored?
- How can we use references to objects in a mathematical knowledge base?
- How do MathWeb agents specify a valid context?

An answer to the last question could be the formal specification of *valid KQML conversations* between MathWeb agents. Labrou has already specified basic KQML conversations in [Lab96] in PROLOG. He suggested that these simple conversations should serve as a basis for more complex ones. The specification of valid conversations requires a common language which is understood by all MathWeb agents. KQML itself could build a basis for such a language, since conversations can be more or less described by an ordered list of schematic KQML messages.

Example 4.1: In the following, we suppose that an ATP agent offers an abstract mathematical service called `prove` which tries to find a proof for a given problem. As described in the last section, we define `prove` as a new OpenMath symbol in the CD `reasys`. The service `prove` can be given some proof assumptions. A specification of valid conversations for `prove` could look like the following:

prove-conv([`tell`(*Sender*, *Receiver*, Openmath, *Formula*)]*
`ask-one`(*Sender*, *Receiver*, Openmath, `prove`(*Formula* , *Result*))
[`tell`(*Receiver*, *Sender*, Openmath, `prove`(*Formula*, *Proof*)) |
`untell`(*Sender*, *Receiver*, Openmath, `prove`(*Formula* , *Result*)) |
`sorry`(*Sender*, *Receiver*, _, _)])

Which means that every MathWeb agent *Sender* who wants to use the `prove` service can first send an arbitrary nonnegative number of facts to the *Receiver* agent, where a fact can be any OpenMath formula. Then *Sender* agent should send exactly one 'ask-one' request with the OpenMath formula to prove and with an OpenMath variable *Result*. Finally, the *Receiver* sends either a tell message containing in which the variable *Result* is replaced by the proof *Proof* or an untell message, if no proof could be found.

[7] These preconditions strongly influence future computations, i.e. the term $\sqrt{x^2}$ can be simplified to x, if it is `assumed` that $x > 0$.

Every abstract mathematical service should provide at least one specification of valid conversations for the service. Each MATHWEB agent who offers an abstract service commits to these conversations, i.e. he must accept every valid conversation for that service. This does not imply the acceptance of a concrete reasoning task, since the agent can still reply with a `sorry` message.

Abstraction in Mathematical Communication. Talking about mathematics can be a very complex task. One means to reduce complexity in general is *abstraction*. Humans perform very well in abstraction and use it very often in their communication. Also in the communication between mathematicians, abstraction can be found. For instance, typical statements of a mathematics professor sound like "*Using the fundamental theorem of algebra, we can infer that ...*", and *not* like "*Using the fundamental theorem of algebra, which states that in the field* $\mathbb{C}$ *of complex numbers... we can infer that ...*". This is due to the fact, that the speaker assumes that details of the fundamental theorem are known to its students. If a student does not know these details, she has to ask her neighbor or look it up in a book.

The main question is, whether we can adopt this human behavior and how we can use some form for abstraction in mathematical communications between MATHWEB agents. In other words: how we can build up contexts of conversations and refer to knowledge in this contexts? What can agents do, if they don't understand a "mathematical statement", because they do not have the expected knowledge? One answer to this question could be to query a mathematical knowledge base, such as MBASE. Thus, the problem of *reference* also affects mathematical knowledge bases. Mathematical knowledge units (e.g. definitions or theorems) are typically stored in MBASE. So it must provide the references to these knowledge units. Additionally, the content language we use for MATHWEB agents must provide the means to handle these references.

4.3 Coordination of MathWeb Agents

We have already mentioned in section 3.3 that sole communication with an agent communication language does usually not lead to a coherent behavior of a multi-agent system. The actions of agents must be coordinated dynamically given a problem by hand. We suppose that MATHWEB agents are benevolent reasoning entities and thus focus on cooperation techniques. One of the central questions is which of the numerous cooperation techniques developed in DAI research are appropriate for MATHWEB agent systems. We try to give a first answer to this question:

Organizational Structuring. MATHWEB agents build a web of heterogeneous reasoning and computation services provided by ATPS, CASs, constraint solver, proof planners, and many other systems. Additionally, MATHWEB agents should perform decentralized distributed mathematical reasoning. This is why cooperation techniques which rely on a fixed organizational structure, e.g. a blackboard architecture, might not be appropriate as a general coordination technique for MATHWEB agents.

However, the blackboard approach can be suitable for some special areas of automated reasoning. In fact, blackboard coordination has recently been applied successfully in the Ω-ANTS approach which provides a command suggestion mechanism for ΩMEGA (see [BS00]). Ω-ANTS' agents are homogeneous and small grained since every agent represents an inference rule or a tactic of ΩMEGA. All Ω-ANTS agents work on a central proof data structure (PDS) which builds the common domain understanding

of the agents. Therefore, the command suggestion mechanism is a perfect application domain for a blackboard architecture (cf. section 3.3).

Multi-agent Planning. In recent years, the ΩMEGA group has made significant progress in proof planning, for instance, in the Limit-domain [Mel97] and in group theory. The latest development is MULTI [MM00], a multi-strategy planner. Up to now, proof planning in ΩMEGA is totally serialized, i.e. at every time the proof planner only plans one specific subgoal. Also in MULTI different planning strategies can only be applied in sequential order. A consequent step for a further improvement of proof planning is the introduction of distribution and parallelism. In a *distributed proof planning* environment, different local planners could concurrently plan with different strategies on the same or different subgoals in order to increase the likelihood of success and to minimize the total planning time.

Sometimes, subgoals in proof planning strongly interact. Typically, certain constraints must be fulfilled by the partial plans for the subgoals[8]. In distributed proof planning these constraints would be spread over the local plans. Therfore a mechanism to insure global consistency of local plans must be developed. Multi-agent planning [FI98] offers the right means for this task and for the coordination of MATHWEB planning agents. But, as mentioned in section 3.3, multi-agent planning requires huge amounts of communication in order to combine the local plans to a single global plan. Thus, the application of this technique should be restricted to subtasks who strongly interact and therfore need the check of global consistency of partial solutions. When the subtasks do not interact, other coordination techniques, such as the contract net protocol should be more appropriate.

Also FISHER and IRELAND proposed the CNP as a coordination technique for proof planning. In [FI98] they presented first ideas on how to bring parallelism into inductive proof planning with CLAM [BvHHS90]. They suggested to assign the different proof steps of a typical inductive proof in CLAM (base-case, rippling, and fertilize) to different planning agents. Having the right communication languages multi-agent proof planning can also include other proof planners like CLAM that are specialized on certain domains (e.g. induction).

Contracting. The contract net protocol is one of the most promising candidates for the coordination of MATHWEB agents in order to perform tasks that do not interact very much. For instance, the CNP could be applied to computation tasks, e.g. the simplification of terms and formulas, or to proving tasks, e.g. proof planning for subgoals without meta-variables or first order problems which can be sent to ATPs.

The bidding process in the CNP allows the contractors to evaluate their local resources available in order to give an according bid. Thus, the CNP can help to realize decentralized load-balancing in a very natural way. But, more often than not, the amount of computational resources available to an agent is not decisive for its success in solving a task. In fact, MATHWEB agents can be seen as specialists in a specific domain, e.g. some reasoners perform well for inductive proofs, others for equational reasoning. Therefore, the performance of an agents is highly dependent on the task to solve. Consequently, the announcement of a task should provide the contractors with a task specification that goes far beyond the specification of abstract mathematical

[8]While planning existential proofs, for instance, meta-variables are introduced as place holders for existentially quantified variables all over the proof plan. These meta-variables must be instantiated by values that must fulfill certain constraints.

services described in section 4.2.2. Additional to the mathematical service requested, the concrete instance of the service must be analyzed.

Example 4.2: We suppose that a CNP manager wants to announce a task `prove`($\forall n.n \in \mathbb{N} \Rightarrow P(n)$), where $P(n)$ is some property over the natural numbers. The analysis of the problem instance, namely the formula $\forall n.n \in \mathbb{N} \Rightarrow P(n)$, would be a valuable source of information for a MathWeb agent specialized on inductive proofs.

However, these are only first ideas, and at the moment it is not clear, whether the contractors or the manager should perform the analysis of the task. The latter would be in the tradition of the classical CNP where the manager only announces an abstract specification of the task rather than the task itself. This is reasonable because, typically, a complete description of a task is very big (cf. Example 2.2).

5 Summary and Work Plan

We argue that some essential features of distributed automated reasoning, such as abstract mathematical services, a decentralization of the problem solving process, and the automatic coordination of reasoning systems, are not met by any of the existing architectures. We propose to apply the agent-oriented programming paradigm to the MathWeb-SB in order to meet these demands. The proposed research can be divided in three main phases:

1. In a first phase we will build the infrastructural foundation for the research. First, we develop and implement the agent shell described in section 4.1. Then, we will encapsulate the reasoning systems integrated in the MathWeb-SB into this agent shell and enable the resulting MathWeb agents to communicate via KQML. This requires the implementation at least of a large part of the KQML specification given by Labrou (cf. section 3.2). MathWeb brokers will be extended in order to perform the functionality of KQML facilitators. The work of this phase will allow us to gain some first experience with MathWeb agents and with KQML conversations.

2. In the second phase, we are going to extend our agent-shell to full MathWeb agents that keep track of the available computational resources and adapt their behavior respectively. Building on this we will develop a distributed version of the ΩMEGA system and its multi-strategy proof planner. We also plan to integrate other proof planning systems, such as the CLAM system into our distributed proof planning architecture.

3. In a third phase we are going to investigate the applicability of the coordination protocols described above and will implement promising protocols. We are going to evaluate the performance of societies of MathWeb agents w.r.t. the different coordination techniques.

5.1 Work Plan

The research is supposed to be done within a period of 3 1/2 years (42 months). For the implementation of the infrastructure in the first phase we estimate 9 months. Since the second phase involves much implementation effort and the development of a reasonable resource management mechanism, we rate 12 months for it. Also phase is very implementation intensive and requires intensive case studies and evaluation processes, so we rate also 12 months for this phase. For writing down the PhD thesis we estimate 9 months.

References

[ABI+96] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[AD93] J. Avenhaus and J. Denzinger. Distributing equational theorem proving. In *Proc. RTA'93, Montreal, LNCS 690*, pages 62–76, 1993.

[AINP90] P. B. Andrews, S. Issar, D. Nesmith, and F. Pfennig. The TPS theorem proving system. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction (CADE–10)*, volume 449 of *LNAI*, pages 641–642, Kaiserslautern, Deutschland, 1990. Springer Verlag, Berlin.

[AZ00] Alessandro Armando and Daniele Zini. Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture. In A. Poggi, editor, *Proceedings of the AI*IA-TABOO Joint Workshop 'From Objects to Agents: Evolutionary Trends of Software Systems'*, Parma, Italy, May 2000.

[BBK99] Patrick Blackburn, Johan Bos, and Michael Kohlhase. Automated reasoning for computational semantics. submitted, 1999.

[BCF+97] 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 William McCune, editor, *Proc. of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

[BG88] Allan H. Bond and Les Gasser, editors. *Readings in Distributed Artificial Intelligence*, chapter An Analysis of Problems and Research in DAI, pages 3–35. Morgan Kaufmann Publishers, San Mateo, California, 1988.

[BM92] S. Bussmann and H. J. Müller. A Negotiation Framework for Cooperating Agents. In *Proc. of the 2nd Workshop on Cooperating Knowledge Based Systems*. Keele, GB, September 1992.

[Bon00] Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):223–257, 2000. Published in February 2001.

[BPSM97] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation PR-xml-971208, World Wide Web Consortium, 1997. Available at `http://www.w3.org/TR/PR-xml.html`.

[BS98] Christoph Benzmüller and Volker Sorge. A blackboard architecture for guiding interactive proofs. In Fausto Giunchiglia, editor, *Artificial Intelligence: Methodology, Systems and Applications*, number 1480 in LNAI, pages 102–114, Sozopol, Bulgaria, 1998. Springer Verlag.

[BS00] Christoph Benzmüller and Volker Sorge. Oants – an open approach at combining interactive and automated theorem proving. In M. Kerber and

M. Kohlhase, editors, *Proc. of the Calculemus Symposium 2000*, St. Andrews, UK, 6–7 August 2000. AK Peters, New York, NY, USA.

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proceedings* 9th *International Conference on Automated Deduction (CADE-9)*, volume 310 of *LNCS*, pages 111–120, Argonne, 1988. Springer Verlag, New York.

[BvHHS90] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In Mark Stickel, editor, *Proc. of the 10th Conference on Automated Deduction*, number 449 in LNCS, pages 647 – 648, Kaiserslautern, Germany, 1990.

[BW94] Russell Beale and Andrew Wood. Agent-based interaction. In *Proceedings of HCI'94*, August 1994. to appear.

[CC98] Olga Caprotti and Arjeh M. Cohen. Draft of the Open Math standard. The Open Math Society, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.

[CH97] J. Calmet and K. Homann. Towards the Mathematical Software Bus. *Journal of Theoretical Computer Science*, 187(1–2):221–230, 1997.

[CL81] D. D. Corkill and V. R. Lesser. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):81–96, 1981.

[CMS83] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. In *IJCAI-83*, pages 767–770, 1983.

[DCN+00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The prosper toolkit. In *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS-2000*, LNCS, Berlin, Germany, 2000. Springer Verlag.

[DD98] Jörg Denzinger and Ingo Dahn. Cooperating theorem provers. In Wolfgang Bibel and Peter Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume 2, pages 483–416. Kluwer, 1998.

[Den93] J. Denzinger. *Teamwork: A method to design distributed knowledge based theorem provers.* PhD thesis, Universität Kaiserslautern, 1993. (in german).

[DS83] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63 – 109, 1983.

[FHJ+99] Andreas Franke, Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, and Volker Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 5:156–187, 1999.

[FI98] Michael Fisher and Andrew Ireland. Towards a science of reasoning through societies of proof planning agents. In Jörg Denzinger, Michel Kohlhase, and Bruce Spencer, editors, *CADE-15 Workshop "Using AI Methods in Deduction"*, pages 33–42, 1998.

[fIPA] Foundation for Intelligent Physical Agents. The fipa 2000 specifications. http://www.fipa.org/.

[FK99] Andreas Franke and Michael Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proceedings of the 16th Conference on Automated Deduction*, number 1632 in LNAI, pages 217–221. Springer Verlag, 1999.

[FMF92] T. Finin, D. McKay, and R. Fritzon. An overview of kqml: A knowledge query and manipulation language. Technical report, University of Maryland, CS Department, 1992.

[Fre79] G. Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879.

[Gea92] M. Genesereth and R. Fikes et al. Knowledge Interchange Format: Version 3.0 Reference Manual. Technical report, Computer Science Department, Stanford University, 1992.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte der Mathematischen Physik*, 38:173–198, 1931.

[Groa] The Object Management Group. The common object request broker architecture. http://www.corba.org/.

[grob] The Oz group. The mozart programming system. http://www.mozart-oz.org/.

[G.W99] G.Weiss, editor. *Multiagent Sytems: a modern Approach to Distributed Artificial Intelligence.* MIT Press, 1999.

[HC96] K. Homann and J. Calmet. Structures for Symbolic Mathematical Reasoning and Computation. In Jacques Calmet and Carla Limogelli, editors, *Design and Implementation of Symbolic Computation Systems, DISCO'96*, number 1128 in LNCS, pages 216–227, Karlsruhe, Germany, 1996. Springer Verlag.

[Hil27] David Hilbert. Die Grundlagen der Mathematik. In *Abhandlungen aus dem mathematischen Seminar der Hamburgischen Universität 6*, pages 65–85, 1927.

[HL99] M.N. Huhns and L.M.Stephens. Multiagent Systems and Societies of Agents. In G.Weiss [G.W99], chapter 2, pages 79–120.

[HLS+96] Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Verification support environment. *High Integrity Systems*, 1(6), 1996.

[HR85] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[HS94] M. Huhns and M.P. Singh. Ckbs-94 tutorial: Distributed artificial intelligence for information systems. Technical report, Dake Centre, University of Keele, England, 1994.

[IM98] P. Ion and R. Miner. Mathematical Markup Language (MathML) 1.0 specification. W3C Recommendation REC-MathML-19980407, World Wide Web Consortium, 1998. Available at `http://www.w3.org/TR/REC-MathML/`.

[JSW98] N.R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7 – 38, 1998.

[KKS98] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[KN00] Alexander Koller and Joachim Niehren. Constraint programming in computational linguistics. In D. Barker-Plummer, D. Beaver, J. van Benthem, and P. Scotto di Luzio, editors, *Proceedings of the eight CSLI Workshop on Logic Language and Computation*. CSLI Press, 2000. To appear.

[Koh] Michael Kohlhase. The OMDoc repository. Internet page at `http://www.mathweb.org/omdoc`.

[Koh00] Michael Kohlhase. OMDoc: An open markup format for mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. `http://www.mathweb.org/omdoc`.

[KvM91] T. Kreifelts and F. v. Martial. A negotiation framework for autonomous agents. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I. 2*. North-Holland, 1991.

[Lab96] Yannis Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland, Baltimore County, 1996.

[LR57] R. D. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey*. John Wiley and Sons, 1957.

[Mel97] Erica Melis. Proof planning limit theorems. Seki Report SR-97-08, Fachbereich Informatik, Universität des Saarlandes, 1997.

[Mel00] E. Melis. The 'interactive textbook' project. In David McAllester, editor, *Proc. of CADE workshop on Deduction and Education; CADE 2000*, number 1831 in LNAI. Springer Verlag, 2000.

[MM00] Erica Melis and Andreas Meier. Proof Planning with Multiple Strategies. In J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, and Y. Sagivand P. Stuckey, editors, *First International Conference on Computational Logic (CL-2000)*, volume 1861 of *LNAI*, pages 644–659, London, UK, 2000. Springer-Verlag.

[MMZ00] E. Melis, T. Müller, and J. Zimmer. Extensions of Constraint Solving for Proof Planning. In W. Horn, editor, *Proceedings of 14th European Conference of Artificial Intelligence (ECAI 2000)*, Amsterdam, 8.–12. August 2000. IOS Press.

[MS99] E. Melis and J.H. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, November 1999.

[MS00] Andreas Meier and Volker Sorge. Exploring properties of residue classes. In M. Kerber and M. Kohlhase, editors, *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*, 2000.

[NLJ96] H. Nwana, L. Lee, and N. Jennings. Coordination in software agent systems. *BT Technology Journal*, 14(4):79–88, 1996.

[Par87] H.V.D. Parunak. Manufacturing experience with the contract net. In M.N. Huhns, editor, *Distributed Artificial Intelligence*, pages 285 – 310. Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence — A Modern Approach.* Prentice–Hall, Englewood Cliffs, 1995.

[Rob65] J. A. Robinson. A Machine Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12:23–41, 1965.

[San93] T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. of the 12th International Workshop on Distributed Artificial Intelligence*, pages 295–308, Hidden Valley, Pennsylvania, May 1993.

[Sea69] J. R. Searle. *Speech Acts.* Cambridge University Press, 1969.

[Sho91] Y. Shoham. Agent-oriented programming. In *Proc. of the 11th International Workshop on DAI*, pages 345–353, 1991.

[SHS98] M. Kohlhase S. Hess, Ch. Jung and V. Sorge. An implementation of distributed mathematical services. In Arjeh Cohen and Henk Barendregt, editors, *6th CALCULEMUS and TYPES Workshop*, Eindhoven, The Netherlands, July 13–15 1998.

[Sie96] Jon Siegel. *CORBA: Fundamentals and Programming.* John Wiley & Sons, Inc., 1996.

[SKM99] J.H. Siekmann, M. Kohlhase, and E. Melis. Omega - a mathematical assistant system. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, pages –. Amsterdam University Press, 1999. published on CDROM and http://www.wins.uva.nl/ j50/cdrom/.

[Smi80] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *IEEE Transaction on Computers*, number 12 in C-29, pages 1104–1113, 1980.

[Smo95] G. Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer-Verlag, Berlin, 1995.

[Ste97] D. Steiner. An Overview of FIPA'97, 1997. Available at `http://drogo.cselt.stet.it/fipa/fipapres1.zip`.

[Syc89] K. P. Sycara. Multiagent compromise via negotiation. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 119–137. Morgan Kaufmann, San Mateo, California, 1989.

[WJ95] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice, 1995. Submitted to Knowledge Engineering Review.

[Zim00] Jürgen Zimmer. Constraintlösen für Beweisplanung. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2000.