Towards the Mathematics Software Bus

Jacques Calmet a,*, Karsten Homann b

a Universität Karlsruhe, Institut für Algorithmen und Kognitive Systeme, Am Fasanengarten 5, 76131 Karlsruhe, Germany
b Siemens AG, Private Communication Systems, Mobile Terminals, Hofmannstrasse 51, 81359 Munich, Germany

Abstract

The Mathematics Software Bus is a software environment for combining heterogeneous systems performing any kind of mathematical computation. Such an environment will provide combinations of graphics, editing and computation tools through interfaces to already existing powerful software by flexible and powerful semantically integration.

Communication and cooperation mechanisms for logical and symbolic computation systems enable to study and solve new classes of problems and to perform efficient computation in mathematics through cooperating specialized packages.

We give an overview on the need for cooperation in solving mathematical problems and illustrate the advantages by several well-known examples. The needs and requirements for the Mathematics Software Bus and its architecture are demonstrated through some implementations of powerful interfaces between mathematical services.

Keywords: Software environments; Symbolic mathematical computing

1. Introduction

The research towards frontiers of combining and integrating systems has been initiated in many areas. The diversity of potential system components of complex systems enclose, for example, models, databases, numerical analyses, graphics, rule bases, computational resources, and vary from the earliest conceptual design and analysis programs to much more detailed and powerful models in the lifecycle of software [2].

The concern about the need for developing a better and formal approach to integrating this diversity of computational resources in suitable software architectures stimulated the studies of abstract methods of combining theories, computational paradigms, algorithms, and other symbolic processing systems.

* Corresponding author. E-mail: calmet@ira.uka.de.

0304-3975/97/$17.00 © 1997 — Elsevier Science B.V. All rights reserved
PII S0304-3975(97)00066-2
For instance, the integration of theorem proving and symbolic mathematical computing has recently emerged from prototype extensions of single systems to the study of environments with interaction among distributed systems. An overview of recent well-known projects on such cooperations is given in the references cited in [3]. However, there are no common architectures, languages, protocols, or standards for such interfaces.

Communication and cooperation mechanisms for logical and symbolic computation systems enable to study and solve new classes of problems and to perform efficient computation through cooperating specialized packages. On the one hand, computer algebra systems (CAS) offer an extensive collection of efficient mathematical algorithms which could improve the efficiency of theorem proving systems (TPS). On the other hand, they ignore Artificial Intelligence methods (e.g. theorem proving, planning of proofs and computations, machine learning) and their capabilities, e.g. verification of properties of mathematical objects using a TPS.

The classification of communication and cooperation methods for logical and symbolic computation systems [7] provides a general framework for methodologies for combining mathematical services [14] and their characteristics, capabilities, requirements, and differences. The advantages of combining systems performing any kind of mathematical computation (mathematical services - M9') are improved expressive power and more powerful inference capabilities. There are various applications for composing those systems, like multi-logic provers, hardware and software verification, proofs with arithmetics and constraints and program transformations.

However, we believe that there can be no one language or mathematical model which is adequate to formalize all objects in design and analysis of mathematical challenges. Our approach allows to include techniques to put mathematical services together, and also provides the primitives of additional mechanisms and infrastructure to select and combine different levels of cooperation. This mechanism has been studied by [18] which introduces integration systems and wrappings.

To allow for integrating already existing mathematical services, the communication and cooperation architecture has been designed as a software bus [6]. Another suitable approach of Multi-User Domains as Interaction Spaces has been presented by [18]. However, the challenge of the Mathematics Software Bus is to enforce a sound common semantics. This is achieved by specifying what a mathematical service is and also through semantically sound definitions of what the operations of linking different pieces of software are. Several results are reported on in several of our papers cited in the references and mainly in [13], for example, the behaviour of objects and control of object distribution among the services. To our knowledge this is the first attempt when referring to both computational and deduction systems. This position paper discusses our hopes and our progress in creating interfaces of services to a software bus for doing mathematics while the corresponding formal semantics of the underlying reasoning and computation theories and services have been introduced in [14].

The paper is organized as follows. Section 2 gives an overview about the need for cooperation in solving mathematical problems. The advantages are illustrated by
several well-known examples. The needs and requirements for the software bus are part of Section 3 and the architecture is given in Section 4.

2. The need for communication and cooperation in symbolic mathematical computing

Problem solving in mathematics often requires the application of both procedural algebraic knowledge (algorithms) and deductive knowledge (theorems). Some of the advantages of combining logical and symbolic computation systems are improved expressive power, more powerful inference capabilities, the introduction of mathematical theories and algorithms, in particular, real numbers and polynomials, into provers, as well as providing logical languages and justifications to symbolic calculators. There are various applications for composing those systems, like multi-logic provers, hardware and software verification, proofs with arithmetics and constraints, program transformations. A complete overview on communication and cooperation in symbolic mathematical computation can be found in [7, 13].

There is a lack of software environments, languages and standards for interfaces between systems for mathematical computation. The reasons are manyfold: (i) CAS and TPS are designed, implemented and validated as stand-alone systems, (ii) many systems are copyrighted and allow neither communication nor external access to internal methods, (iii) they do not provide interfacing.

Several communication and cooperation methods have already been examined. The basic level of cooperation is just to exchange mathematical information. To enable mathematicians, TPS or CAS to pass proofs, theorems, functions, algorithms or any kind of mathematical objects offline by electronic mail, cut & paste or ftp requires communication in terms of a common language. Open Mechanized Reasoning Systems [11] and OPENMATH [1] introduce general languages suitable for specifying and communicating mathematical objects in theorem proving and symbolic mathematical computing, respectively, which can be composed [14].

Higher levels of online cooperation can be achieved by adding links to interactive tools. The interfaces between HOL and MAPLE [12] and ISABELLE and MAPLE [3] introduce the powerful arithmetics of a computer algebra system into a tactical theorem prover to reason about numbers or polynomials much more efficiently. MAPLE acts as a slave to the prover which controls external calls by evaluation tactics. Jackson [15] presents an interaction to provide expressive algebra of constructive type theory in computer algebra. The theorem prover NUPRL is an algebraic oracle to the CAS Weyl. ANALYTICA [8] is an example for cooperation within the language of another system. It is written in the Mathematica environment and can solve sophisticated problems in elementary analysis.

CAS/π is a sophisticated example of a powerful graphical system-independent common user interface [16]. It was designed so that expert users can set up connections to alternative CAS or visualization tools easily and at runtime.
An architecture for distributed theorem proving is given in [10]. TPS compete and then cooperate using completion in pure equational logic using team work.

The advantage of distribution is to profit from heuristics of several systems to reduce the typically immense search spaces. An extension to distributed proof planning is to integrate the CAS SAPPER in the interactive proof development environment $\Omega$-MKRP, not at the system level but at the level of proofs [17]. Such an architecture is one of the first steps towards the cooperation of CAS and TPS without trusting external computations. The result is a correct proof which is carried out by the aid of external computations.

3. Technical requirements

To provide interfaces between existing mathematical services requires an open software architecture. Among the aspects that must be further investigated are messaging and communication facilities, information modelling, infrastructure, mathematical service modelling application and knowledge base integration, development and management tools. Some of these requirements are not specific to mathematical knowledge and have been studied in different applications, for example, an Information Bus and Enterprise Toolkit in manufacturing, construction, and banking sectors [20].

There is a lack of software environments, languages and standards for interfaces between systems for mathematical computation. The reasons are manyfold: (i) CAS and TPS are designed, implemented and validated as stand-alone systems, (ii) many systems are copyrighted and allow neither communication nor external access to internal methods, (iii) they do not provide interfacing.

In the rest of this section we discuss some of the above aspects by instantiating common engineering techniques to mathematical services: communication languages, information exchange, and common knowledge representation. The resulting architecture is discussed in the following section.

3.1. Communication languages

A communication language defines how mathematical information can be exchanged among services. It must be recognized by each system in order to translate the information into their internal representation. Appropriate languages can either be the input language or internal encoding of one of the involved systems or standardized communication languages.

Several communication languages for interfaces between software systems exchanging mathematical information have been developed. We propose an extension of OPENMATH [1], which classifies system combinations according to the framework given in the basic OPENMATH model. Fig. 1 illustrates an interface which is part of any mathematical service. Some applications may not distinguish between some of the levels.
The communication is not implemented as the input language to one of the involved systems but as an interface compiling the service specific representation into a standardized encoding. This encoding is either a stream of bytes or an extended Lisp-like representation suitable for transmission via files, cut & paste, email, ftp and broadcasting like Unix sockets [7].

3.2. Information exchange

Cooperation among several software systems can be achieved with indirect, unidirectional and bidirectional communication. According to the flow of mathematical information, several exchange paradigms have to be integrated to the software bus. Some of these are illustrated in Fig. 2.

Although there are no direct links between the services with indirect communication, interaction is possible if both systems can communicate with a common user interface, central unit, mediator or evaluator. Such an interface provides links to some MSF. A user can access the systems and can apply (symbolic or numerical) algorithms or theorems to solve a given problem, depending on the class of the problem. Such a simple type of interaction allows already the use of arbitrary CAS and ATP. However, the systems do not interact directly and a user must be familiar with both systems.

To manage the communication and to hide the control from the user interface leads to an architecture with common evaluator or central control. The evaluator controls the selection of the modules by meta-knowledge on all functions and predicates. It also controls the application of algebraic algorithms and exchange of data and theorems in the MSF. The mathematical knowledge is represented separately in each module. The Mathematics Software Bus generalizes the Central Control environment [9] which is a typical representative for this architecture. The tools are mainly independent: they can perform their tasks without the help of other tools.
Unidirectional links can most often be found when communicating with input or output devices like math editors, visualization tools, graphical interfaces, SGML, in case of master/slave cooperation, or subpackages.

3.3. Common knowledge representation

Many applications require several MSP to share their knowledge about mathematical objects. In many cases, communicating this information is neither efficient nor practical, because it may not be explicitly known which knowledge is required.

Some cooperation mechanisms obviously benefit from sharing their knowledge, i.e. communication with subpackages or direct function calls in foreign packages like Analytica [8]. The Mathematics Software Bus may include a knowledge representation system suitable for representing the common knowledge. Both architectures are illustrated in Fig. 3.

Recent communication environments are not restricted only to exchange of function calls, theorems, numerical data, polynomials or basic mathematical information. For
example, the software bus provides the exchange of mathematical objects with a defined semantics derived from its associated lexicons. However, there are no protocols to provide meta-knowledge about the systems algorithms or type information about their arguments.

To represent explicitly the mathematical information embedded in CAS requires to introduce the representation of meta-information, e.g. in terms of schemata [13]. Different schemata contain this knowledge as type schemata and algorithm schemata.

4. The Mathematics Software Bus architecture

The first environments providing bidirectional links have been studied recently [12, 3]. Such a communication requires to exchange common mathematical objects or relies on a common knowledge representation. At any step, arbitrary combinations of algorithms and theorems should be applied to solve a given problem. This combines the advantages of all involved mathematical services.

The Mathematics Software Bus generalizes uni- and bidirectional communication and common knowledge representation. Fig. 4 illustrates this open architecture combining heterogeneous mathematical services. The highlighted connection between MAPLE and ISABELLE is a generalization of one of our bidirectional interfaces [3].

The development of an open software architecture requires the decomposition of system into subsystems, the distribution of control and responsibility, and the development of the components and their connections or means of communication. Ref. [5] gives references to architectural styles.

We discussed a vocabulary of the basic design elements [13, 1] in the previous section. A set of configurations constrain how services and their interfaces may be configured. The semantics defines the meaning of a suitably configured bus and allow to analyse a configuration.

Fig. 4. Software bus of mathematical services.
Example

We implemented several interfaces between mathematical services, e.g. ISABELLE and MAPLE, DTP and MAGMA, IMPS and MAPLE. As an example, the rest of this section illustrates the highlighted connection of Fig. 4.

We designed and implemented an interface to the tactical theorem prover ISABELLE to be connected to the Mathematics Software Bus. Beside providing the distribution of control and responsibility, we implemented this interface by extending the simplifier of ISABELLE [19] by new kinds of rules to call external functions over the bus. The choice of MAPLE as computational service was arbitrary except some syntax translations when interfacing the CAS.

The semantics of an interactive proof involves computation structures [14] with only one algorithm application. As an example, the inductive proof of

$$\forall n \in \mathbb{N} : 5 \leq n \Rightarrow n^5 \leq 5^n$$

expands all of the products in the induction step by communication of the corresponding expand expression to MAPLE. This access to the bus is hidden to the user and is performed inside interfacing tactics and extended simplification sets [3].

- by (res_inst_tac [("P",
= "%y. expand((x + 1) \^ 5) \leq y")]) expandE 1);
n \^ 5 <= 5 \^ n
1. \!\!x. [\mid x : \text{Nat}; 5 \leq x; x \^ 5 \leq 5 \^ x \mid] =>
   expand((x + 1) \^ 5) \leq expand(5 \^ (x + 1))
- by (asm_simp_tac Nat_simplify_ss 1);
n \^ 5 <= 5 \^ n
1. \!\!x. [\mid x : \text{Nat}; 5 <= x; x \^ 5 <= 5 \^ x \mid] =>
   x \^ 5 + 5 \* x \^ 4 + 10 \* x \^ 3 +
   10 \* x \^ 2 + 5 \* x + 1 <= 5 \* 5 \^ x

Three basic functions enable access to the bus: init_session, exchange_service, and terminate_session. At initialization, ISABELLE announces availability to all of the connected services. terminate_session issues BREAK commands to all active internal and external ISABELLE connections. One single function exchange_service allows external function calls. This is the implementation of an unidirectional link, as the TPS is an interactive master service with direct user control and MAPLE accepts external requests through its bus interface. It is a first step towards the implementation of the Mathematics Software Bus.

5. Conclusions

Cooperation by distributing tasks between mathematical services is a subject of ongoing research. Among the arising problems is the black box behaviour of almost any
current system. To plan and control such environments requires to represent metaknowledge in local or global bridges or supervisors. The significance of the software bus architecture has been illustrated by its capability to integrate existing powerful mathematics software by plug & play.¹

Among the work in progress is the design of an intelligent assistant – an environment built on top of the Mathematics Software Bus, whose semantics allows a consistent treatment of algorithms and theorems. A result of this work is the generalization of our interface between the tactical theorem prover IMPS and MAPLE [13] to the bus. The extension of contexts [6] is another step towards a semantics of software environments performing distributed mathematical problem solving.

References


¹The system kernels remain unchanged and only some additional definitions and initialisations are required for the interface to the bus.


