Formal Aspects of Computing (1999) 11: 326–342
© 1999 BCS

Formal Aspects of Computing

$\mathcal{L}\Omega\mathcal{UI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface

Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet and Volker Sorge

FB Informatik, Universität des Saarlandes, Saarbrücken, Germany

Keywords: Automated deduction; Mathematics; User interface; Agents; Distributed Artificial Intelligence; Human Computer Interaction

Abstract. The capabilities of a automated theorem prover's interface are essential for the effective use of (interactive) proof systems. $\mathcal{L}\Omega\mathcal{UI}$ is the multi-modal interface that combines several features: a graphical display of information in a proof graph, a selective term browser with hypertext facilities, proof and proof plan presentation in natural language, and an editor for adding and maintaining the knowledge base. $\mathcal{L}\Omega\mathcal{UI}$ is realized in an agent-based client-server architecture and implemented in the concurrent constraint programming language OZ.

1. Introduction

The effective use of an interactive theorem proving system depends not least on the capabilities of its user interface. A major problem is the adequate access to the overwhelming amount of information manipulated by these systems. This requires structure-oriented overview facilities, and selective and precise content-oriented display.

The $\mathcal{L}\Omega\mathcal{UI}$ system is a state-of-the-art user interface for ΩMEGA, a proof assistant system for mainstream mathematics based on proof planning [BCF97]. Some of $\mathcal{L}\Omega\mathcal{UI}$'s distinct features are the graphical visualization of the information in proofs, a selective term browser as well as proof and proof plan presentation in natural language. The user can add new information to partial proofs, which helps to understand and guide the proof search. $\mathcal{L}\Omega\mathcal{UI}$ is implemented as a client in the distributed agent architecture MATHWEB [FrK99, FHJ99].

Correspondence and offprint requests to: Prof. Dr Jörg Siekmann, FB 14, Universität des Saarlandes, D-66041 Saarbrücken, Germany. `http://www.ags.uni-sb.de`

We start the paper with the design motivation which influenced the development of the $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ system. In Section 3, various display facilities including graphical and natural language presentations are described. This is followed in Section 4 by an illustration of a number of control mechanisms for the effective use of proof techniques. Section 5 is devoted to the client-server architecture. Finally, we discuss some related work and directions for further work.

2. Design Objectives

In the field of interactive theorem proving, design principles for the development of graphical user interfaces are still relatively rare. Some guidelines are presented in [Eas98]. The design objectives that we have focused on for our interface are:

Multi-Modal Visualization. In any proof state the system should display the proof information to the user at different levels of abstraction and detail and furthermore in different modes (e.g. as the graphical representation of a proof tree, as a linearized proof, or in verbalization mode).

Lean Processing. The interface should work reasonably fast, and its installation in other environments should be possible with minimal effort and storage requirements.

Anticipation. The system should minimize the necessary interaction with the user by suggesting commands and parameters at each proof step. Optimally, the system should be able to do the straightforward steps autonomously.

One principle mentioned in [Eas98] is the guideline that "there should be a number of complementary views of the proof construction and the user should be able to choose to see any number of the views simultaneously". In other words, a **multi-modal visualization** is desirable. Most proof systems however concentrate just on one single view of the proof rather than on alternative presentations. In contrast, $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ provides different and complementary views of a proof such as a graphical display or a linearized proof (see Section 3). The traditional graphical tree representation of the proof is enhanced by dedicated browsers for selected textual information (see Fig. 1) and intensive use is made of hypertext techniques in order to illustrate connections between physically distant, but logically related portions of proofs in both the text-based and the graphical modes. For instance it is easy to inspect a proof line's premises with these links and to return to the starting point by clicking on the corresponding history button in the symbolbar. To add a natural language view of proofs, $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ calls the PROVERB proof presentation system that structures and verbalizes proofs in natural language (see Section 3.3).

The **lean processing** principle has led to a distributed system architecture of $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$/ΩMEGA, where $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ is realized as an autonomous software agent (see Section 5), which can be sent over the Internet as an applet while the ΩMEGA server resides on a dedicated host. Since $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ is an autonomous agent, it maintains its own representation of the proof state and autonomously computes the visualization information by using local computational resources, thus reducing the communication bandwidth to a minimum. Thus the architecture inherits the advantage from two kinds of setup: From one, where the whole deduction system is installed locally on the client machine (local computation) as well as from one, where the logical and graphical computations are centralized on a server the user communicates with, say, by a remote X connection. This enables the realization of the concept of **direct manipulation** [Shn92], which allows for immediate feed-back and a minimal time a user has to wait until an action takes effect. Direct manipulation is supported since $\mathcal{L}\Omega\mathcal{UI}$ can react to many forms of user interaction immediately by manipulating its internal representation of the proof state rather than calling the server.

Anticipation to minimize user interaction, has always been a concern of interactive systems, for instance by disabling commands that are nonsensical (i.e. pre-selecting legal actions) or generating lists of commands that are advisable in a current situation. In $\mathcal{L}\Omega\mathcal{UI}$'s internal representation of the proof state, many interface-related reasoning tasks can be performed without the help of the underlying proof system. For example, $\mathcal{L}\Omega\mathcal{UI}$ supports and complements the agent-based command suggestion mechanism [BeS98a] provided by its host system ΩMEGA.

We shall elaborate on these issues in the following sections.

3. Multi-modal Views

$\mathcal{L}\Omega\mathcal{UI}$'s presentation features are, to a certain extent, influenced by ΩMEGA's central three-dimensional proof data structure $\mathcal{PDS}$, which will be presented in section 3.1. The two subsequent sections discuss the principal proof presentation capabilities in $\mathcal{L}\Omega\mathcal{UI}$: a structural tree visualization with references to terms and inference steps and a natural language display. To start with, consider the following example:[1]

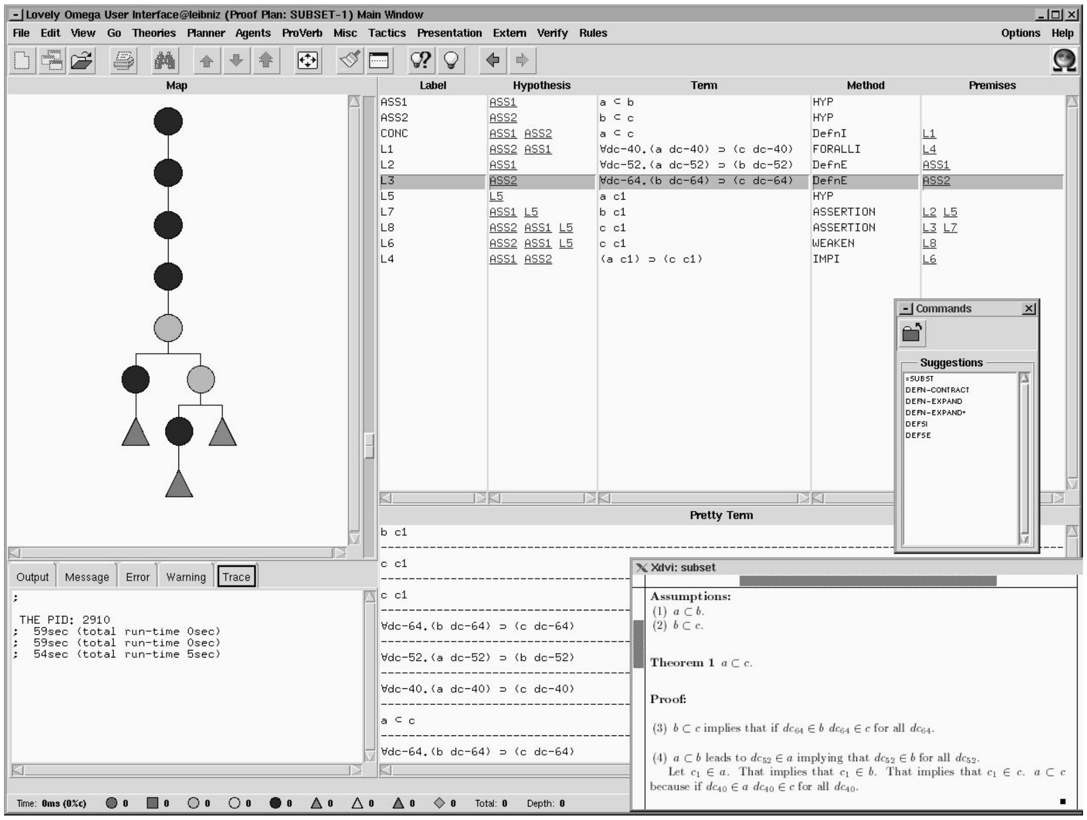
Theorem 1 (Example) Given that $a \subset b$ and $b \subset c$. Then $a \subset c$.

A proof of this theorem can be generated in ΩMEGA in many ways, the easiest is by calling an external reasoner for such a simple problem, e.g. the first-order theorem prover OTTER [McC94], which quickly finds the proof. This external proof is then translated into ΩMEGA's proof format and inserted into the central proof data structure $\mathcal{PDS}$. Now, $\mathcal{L}\Omega\mathcal{UI}$ provides different components to view this data structure (see Fig. 1 for a screen-shot). As in traditional theorem proving systems, $\mathcal{L}\Omega\mathcal{UI}$ can present a proof in a linear text form, in our case as a higher-order variant of Gentzen's natural deduction (ND) calculus (as in the upper right frame in Fig. 1). The formula of the highlighted line is pretty-printed in the term browser (see the lower right frame in Fig. 1). For long proofs, such a presentation lacks transparency and structure. Therefore $\mathcal{L}\Omega\mathcal{UI}$ offers two additional representations:

- as a tree that models the logical dependencies between the different proof lines, (see the upper left frame in Fig. 1),
- as a text in natural language as it would appear in a mathematical textbook (see Xdvi-window in the lower right corner of Fig. 1). Currently, only completed proofs can be presented in natural language.

Furthermore $\mathcal{L}\Omega\mathcal{UI}$ uses hypertext techniques to visualize essential connections, e.g. between the proof lines in the standard linearized proof and the corresponding nodes in the tree representation.

[1] All examples in this paper are chosen for presentation purposes, not as an example of realistic scale.


Lovely Omega User Interface@leibniz (Proof Plan: SUBSET-1) Main Window
File Edit View Go Theories Planner Agents ProVerb Misc Tactics Presentation Extern Verify Rules Options Help
Map
Commands
Suggestions
Pretty Term
Output Message Error Warning Trace
Xdvi: subset

Label	Hypothesis	Term	Method	Premises
ASS1	ASS1	a ⊂ b	HYP	
ASS2	ASS2	b ⊂ c	HYP	
CONC	ASS1 ASS2	a ⊂ c	DefnI	L1
L1	ASS2 ASS1	∀dc-40.(a dc-40) ⊃ (c dc-40)	FORALLI	L4
L2	ASS1	∀dc-52.(a dc-52) ⊃ (b dc-52)	DefnE	ASS1
L3	ASS2	∀dc-64.(b dc-64) ⊃ (c dc-64)	DefnE	ASS2
L5	L5	a c1	HYP	
L7	ASS1 L5	b c1	ASSERTION	L2 L5
L8	ASS2 ASS1 L5	c c1	ASSERTION	L3 L7
L6	ASS2 ASS1 L5	c c1	WEAKEN	L8
L4	ASS1 ASS2	(a c1) ⊃ (c c1)	IMPI	L6

```
*SUBST
DEFN-CONTRACT
DEFN-EXPAND
DEFN-EXPAND*
DEFSI
DEFSE
```

```
b c1
--------------------------------------
c c1
--------------------------------------
c c1
--------------------------------------
∀dc-64.(b dc-64) ⊃ (c dc-64)
--------------------------------------
∀dc-52.(a dc-52) ⊃ (b dc-52)
--------------------------------------
∀dc-40.(a dc-40) ⊃ (c dc-40)
--------------------------------------
a ⊂ c
--------------------------------------
∀dc-64.(b dc-64) ⊃ (c dc-64)
```

```
;

 THE PID: 2910
;   59sec (total run-time 0sec)
;   59sec (total run-time 0sec)
;   54sec (total run-time 5sec)
```

Time: **0ms (0%c)** ● **0** ■ **0** ○ **0** ○ **0** ● **0** ▲ **0** △ **0** ▲ **0** ◇ **0** Total: **0** Depth: **0**

Assumptions:
(1) $a \subset b$.
(2) $b \subset c$.

Theorem 1 $a \subset c$.

Proof:

(3) $b \subset c$ implies that if $dc_{64} \in b$ $dc_{64} \in c$ for all dc_{64}.

(4) $a \subset b$ leads to $dc_{52} \in a$ implying that $dc_{52} \in b$ for all dc_{52}.
Let $c_1 \in a$. That implies that $c_1 \in b$. That implies that $c_1 \in c$. $a \subset c$ because if $dc_{40} \in a$ $dc_{40} \in c$ for all dc_{40}. ■

Fig. 1. The $\mathcal{L}\Omega\mathcal{UI}$ interface presenting the proof for Theorem 1. The standard text presentation is given in the upper right frame, whereas the corresponding proof tree is given in the upper left frame. The term browser in the lower right frame displays the formula of the currently focused proof line/node. The lower left frame provides information on different message streams from the ΩMEGA system. The lower-right Xdvi-window presents the verbalized proof as generated by PROVERB.

The command menu bar on top of the entire frame in Fig. 1 provides access to ΩMEGA's proof tools that are organized in pull-down menus. Icons are used as shortcuts to specific commands or sub-menus. Command suggestions for the next proof step are presented for quick and easy selection in a special suggestion window. Finally, a control window (see the lower left frame in the window displayed in Fig. 1) provides access to the output streams of ΩMEGA's processes. In the following subsection we shall present the details of this visualization and the motivation underlying its design.

3.1. Hierarchical Proof Plan Data Structure

Finding a proof with ΩMEGA can be viewed as a process that interleaves proof planning [Mel98, ChS98], plan execution, and verification, all of which is data-driven by the so-called *Proof Plan Data Structure* ($\mathcal{PDS}$).

This hierarchical data structure represents a (possibly partial) proof at different levels of abstraction, called *proof plans*. Its nodes correspond to steps of the derivation and are justified by methods. Conceptually, each justification represents a proof plan for the corresponding derivation step at a lower level of abstraction (the *expansion* of the justification). It is computed when the method is expanded. A proof plan can be recursively expanded, until a proof at the calculus level has been reached. In ΩMEGA, we keep the original proof plan in the expansion hierarchy of the $\mathcal{PDS}$. Thus, the $\mathcal{PDS}$ makes explicit the hierarchical structure of proof plans and retains it for further applications such as proof explanation or analogical transfer of plans.

When the expansion process is completed, a verifier can check the correctness of the proof. The expansion of macro steps provides the basis for an integration of external reasoning components – such as an automated theorem prover (ATP) or a computer algebra system (CAS) – if each reasoner's results can (on demand) be transformed into a sub-$\mathcal{PDS}$. New pieces can be added to the $\mathcal{PDS}$ by directly calling methods and rules, by inserting facts from a database, or by calling some external reasoner.

$\mathcal{L}\Omega\mathcal{UI}$ supports ΩMEGA's three-dimensional $\mathcal{PDS}$ in many ways. For instance, different layers of the $\mathcal{PDS}$ can be analyzed separately and switching to another layer is supported by context menus for each node. In this sense, $\mathcal{L}\Omega\mathcal{UI}$ implements the philosophy of multi-dimensional representations of proofs within its visualization and control facilities.

3.2. Visualization – Proofs as Tree-like Maps with Associated Content

If the proof information is conveyed in only one mode by the user interface, it can lead to problems:

- Presentation in linear form fails to convey the structure of a proof.
- Presentation in tree form may soon become difficult to survey because of the large annotations associated with each node.

Because of the inadequacies of purely linear or tree formats, a central design decision in $\mathcal{L}\Omega\mathcal{UI}$ was to separate the structure of the proof tree from its content. Consequently, the visualization of a proof in $\mathcal{L}\Omega\mathcal{UI}$ consists of three parts:

- a proof tree in a purely structural form, where the individual nodes convey status information through their shape and colour.
- a linear form of the content of the proof, by individual lines.
- a number of co-reference facilities for the connections within and between the tree and the linear proof visualization forms.

The linear form of the proof display is fairly standard in most of its parts, where each derivation step is presented in one single line. These steps of a derivation usually fit into a reasonably sized window as the associated display demonstrates (see the upper right part of Fig. 1). This may not be the case for entries in the part named "term". Therefore a separate frame selectively displays a single term in full length, which can be activated by clicking on the term of interest in the linear format.

Logical proofs are in general acyclic directed graphs rather than trees, hence the graphical display of such structures poses problems: If a pure tree display is produced by duplicating identical subproofs, the tree may grow very large. If, alternatively, multiple subtrees are displayed only once with pointers to the other positions, these pointers may easily render the visualization confusing and unmanageable. Therefore, $\mathcal{L}\Omega\mathcal{UI}$ represents nodes with multiple predecessors (i.e. subproofs used more than once) as *co-reference* nodes: The subproof is displayed only in one place, and the other occurrences are represented as a special node –

the co-reference node. In a sense, co-reference nodes take the role of a lemma, but a co-reference node is not necessarily promoted into a lemma.

Using this convention, proof graphs can be visualized in $\mathcal{L}\Omega\mathcal{UI}$ as proper trees, where node categories (representing the status of the node in the $\mathcal{PDS}$) are expressed by colour and shape. The shape illustrates the major node category and colour variations express more fine-grained distinctions:

Terminal nodes are represented by *upward pointing triangles*, where *assumptions*, *assertions*, and *hypotheses* are distinguished by their colour (green, yellow, and orange).

Intermediate nodes are represented by *circles*, where *ground*, *expanded*, and *unexpanded* nodes are distinguished by colour (dark blue, bright blue, and light blue).

Open nodes are represented by *squares*. Since there are no further categorical distinctions for open nodes, they have the same colour (red).

Co-reference nodes, which may or may not be terminal nodes, are represented by *downward pointing triangles*, they are uniquely coloured in grey.

Open nodes represent subgoals in the proof which have not been solved yet, i.e. they are subject to further derivations. Intermediate nodes represent the respective level of abstraction in the $\mathcal{PDS}$: Ground nodes are at ΩMEGA's calculus level, i.e. a set of ND rules that is large enough to ensure completeness. The other intermediate nodes represent inference steps at higher levels of abstraction. Expanded nodes are nodes where the expansion to the next lower level was already calculated, but is not displayed. In unexpanded nodes, the expansion has not yet been calculated.

In order to obtain a good view of the proof tree, the user has commands to manipulate the appearance of that tree:

zooming between tree overviews and enlarged tree parts,

scrolling to a desired tree part,

focusing on a subtree by cutting off the remaining tree parts,

abstracting away from details of a subtree derivation by hiding the display of that subtree, which then appears as a double-sized red triangle.[2]

Because of the different forms of proof display, especially tree structure and content, various overview formats are offered for the entire proof, where the references between elements of this overview are very selective and only triggered by explicit user commands. There are four different forms of co-references in $\mathcal{L}\Omega\mathcal{UI}$'s display:

- Co-references *within the linear form*, including the "Pretty Term" frame. Two facilities are offered here. One is activating the "Pretty Term" browser, which is done by clicking on the term of interest. The selected term is then displayed in full length in the "Pretty Term" frame, while the line in which that term appears is highlighted (see Fig. 1). The other facility is for inspecting individual justifications of a derivation, which is achieved by clicking on the premise of interest in a selected line of proof. Again, this line is highlighted.

[2] Note that this subtree abstraction is different from abstraction levels in the $\mathcal{PDS}$.

- Co-references *within the proof tree*. Through this facility the connection of a co-reference node is re-established temporarily. Pointing to a co-reference node leads to the temporary appearance of a line between that node and the node it co-refers to, that is, the root of the subtree representing the subproof hidden behind the co-reference node.
- Co-references *between the linear form and the proof tree*. The connection between structure and content can be established through this facility. Clicking on a node activates a yellow box next to that node in the tree display which contains a label and a term. The referred line in the linear form of the proof is highlighted.
- Co-references *between plain text and the proof tree, the linear form and the menu bars of $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$*. There are currently two co-references of this kind. One is from the node of a proof tree or proof line to a verbalization of the justification of this node in natural language as described in Section 3.3. The second kind of co-reference is from hypertext documents (like the online documentation), where hyperlinks can be used to directly activate $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ commands (see Section 4).

3.3. Proofs in Natural Language

While $\mathcal{L}\Omega\mathcal{U}\mathcal{I}$ cannot read natural-language input yet,[3] it makes use of the PROVERB system [HuF97] to present proofs in natural language. PROVERB employs state-of-the-art techniques of natural language processing and generation.

Like most application-oriented natural language generation systems, PROVERB has a pipelined architecture [Rei94] consisting of three processing phases, each realized by a dedicated component: a macro-planner, a micro-planner, and a surface realizer. The macro-planner linearizes a proof and plans communicative acts by a combination of hierarchical planning and focus-guided navigation. The micro-planner then maps communicative acts and domain concepts into linguistic resources, it paraphrases and aggregates such resources to produce a text structure that contains all necessary syntactic information. The realizer TAG-GEN [KiF95] executes this text structure and generates the surface sentences that are passed on to LaTeX2e. The formatted text is then finally displayed in an Xdvi-window (cf. Fig. 1).

While the underlying architecture is standard for many language generation systems, PROVERB has a number of special features that are particularly useful for presenting mathematical proofs: a *focus mechanism* to control the presentation of proof steps in context, *paraphrasing capabilities* to augment the system's expressiveness, and *aggregation operators* that can be employed to express facts that share some of their referents and predicates.

The *focus mechanism* is inspired by Reichman's theory of discourse [Rei85]. It hypothesizes a set of nested focus spaces which regulate referential accessibility of discourse referents, that is, lemmata, theorems, and logical facts in the domain of mathematical proofs. The focus spaces are used to anticipate whether or not a particular discourse referent in the communicative act considered is in the addressee's focus of attention. This determines for example whether the premises

[3] We are currently working in a collaborative effort within the SFB 378 to read a mathematical text from a text book.

for a derivation step are omitted, explicitly repeated, or implicitly hinted at by the conclusion or the method justifying that step.

The *paraphrasing capabilities* are based on the systematization of Meteer's Text Structure [Met92] that guarantees the compositional expressibility of domain concepts in natural language terms through a hierarchy of semantic categories. For example, depending on the embedding context the logical predicate $para(C1, C2)$ can verbally be expressed as a *quality relation* ("line $C1$ is parallel to $C2$"), as a *process relation* ("line $C1$ parallels $C2$"), or as a *property ascription* ("lines $C1$ and $C2$ are parallel" or "the parallelism of lines $C1$ and $C2$").

Finally, the *aggregation operators* constitute some specific instances of general and linguistically-motivated structure modification operations, such as those found in [DaH93]. Apart from domain-specific, pattern-based optimization rules, there are two sorts of aggregation operators with a general scope in PROVERB that handle *predicate grouping* and *semantic embedding*. The predicate grouping aggregation condenses two assertions with the same predicate into a single one, e.g. $Set(F) \wedge Set(G)$ can compactly be expressed as "F and G are sets". *Semantic embedding* allows the skilful verbalization of one assertion, such that it embeds into another one. For example in $Set(F) \wedge Subset(F, G)$ the verbalization of $Set(F)$ as the noun phrase "the set F" allows this expression to be embedded into "F is a subset of G," yielding "The set F is a subset of G".

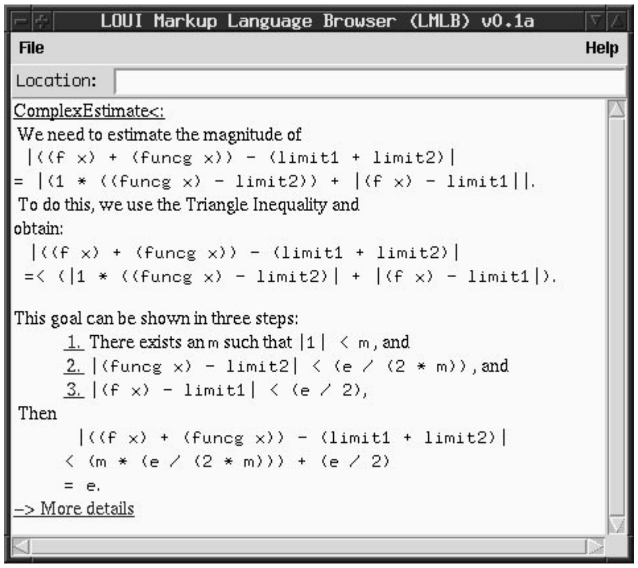
Altogether, implementing the linguistically motivated concepts of focus spaces, paraphrasing, and aggregation into PROVERB significantly contributes to the production of a shorter and better readable proof verbalization in comparison to a direct verbalization of the lines of a proof trace. The presentation of mathematical proofs in natural language by PROVERB can be further improved by taking into account the user's background knowledge and associated reasoning capabilities.

A more recent presentation facility is the natural language presentation at the more abstract level of proof plans [MeL99]. Proof planning is based on reasoning steps similar to those used by mathematicians. It is therefore more natural to generate a verbalization on this level. The communication with the user is facilitated by presenting a verbalization of a method in a hypertext window when the corresponding node of the tree presentation is clicked on. This hypertext presentation of a method offers further links to the verbalization of proofs of subgoals introduced by the method (see Fig. 2). Currently, this local presentation of methods can be aggregated to a global presentation of the whole proof plan, but needs further elaboration.

4. Controlling ΩMEGA

ΩMEGA's main functionalities are available via the structured menu bar in $\mathcal{L}\Omega\mathcal{UI}$'s main window. Its entries partition the conceptually different facilities of the ΩMEGA-system into topics, such as *Theories*, *Extern*, *Planner*, and *Agent*. These provide all commands of the respective conceptual category. Commands may be further grouped into submenus. For example, the topic *Rule* provides different inference rules that define the basic calculus of the ΩMEGA system. These rules are grouped into categories reflecting their logical effect on a proof, for instance elimination or introduction rules.

One important feature of $\mathcal{L}\Omega\mathcal{UI}$ is its dynamic and generic menu extension, i.e. commands are incrementally added at run-time when they are required and



```
ComplexEstimate<:
 We need to estimate the magnitude of
  |((f x) + (funcg x)) - (limit1 + limit2)|
= |(1 * ((funcg x) - limit2)) + |(f x) - limit1||.
 To do this, we use the Triangle Inequality and
obtain:
  |((f x) + (funcg x)) - (limit1 + limit2)|
 =< (|1 * ((funcg x) - limit2)| + |(f x) - limit1|).

This goal can be shown in three steps:
      1. There exists an m such that |1| < m, and
      2. |(funcg x) - limit2| < (e / (2 * m)), and
      3. |(f x) - limit1| < (e / 2),
 Then
        |((f x) + (funcg x)) - (limit1 + limit2)|
      < (m * (e / (2 * m))) + (e / 2)
      = e.
-> More details
```

Fig. 2. Verbalization of a proof planning method.

the user can easily create new menu entries by specifying new commands. This is achieved by defining commands separately from ΩMEGA's core system. Some commands are then loaded initially. Others – for instance, commands that execute domain specific tactics – are loaded only when the appropriate theory (see below) is imported. Thereby a command is always attached to a single menu topic and, if appropriate, to one or several submenus.

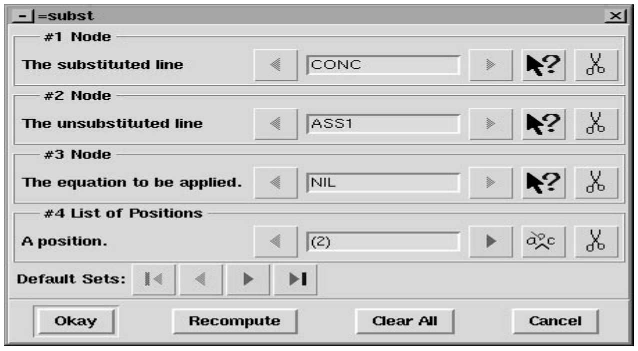
Theories. Mathematical knowledge in ΩMEGA is hierarchically structured with respect to so-called theories, which contain signature information, axioms, definitions, theorems, lemmata, and the basic means to construct proofs, namely rules, tactics, planning methods, and control knowledge for guiding the proof planner.

Each theorem $\mathcal{T}$ has its home theory and therefore a proof of $\mathcal{T}$ can use the theory's signature extensions, axioms, definitions, and lemmata without explicitly introducing them. A simple inheritance mechanism allows the user to incrementally build larger theories.

The user can both use and manage ΩMEGA's knowledge base through $\mathcal{L}\Omega\mathcal{UI}$. In particular, it is possible to load theories or their single components incrementally and separately, browse through available assertions and import them into the active proof. Furthermore, if a theorem has been proved and the proof is verified, it can be stored in its home theory.

Rules and Tactics correspond to inference steps at different levels of abstraction. (We will refer to both rules and tactics with the generic term *inference*). While rules belong to a (low level) logic calculus, tactics are more abstract. They are programs that, when executed, produce a sequence of less abstract inference steps. Generally, each inference is associated with exactly one command which invokes it in a given partial proof.

The hierarchic organization of theories and their incremental importation not only affects their availability for a proof but also $\mathcal{L}\Omega\mathcal{UI}$'s menu structure. Since theories contain rules and tactics, these are also incrementally loaded and each attached command is dynamically appended to the corresponding topic in the menu. Since rules are always defined in ΩMEGA's base theory, they are just sorted


=subst

#1 Node
The substituted line: CONC

#2 Node
The unsubstituted line: ASS1

#3 Node
The equation to be applied.: NIL

#4 List of Positions
A position.: (2)

Default Sets:

Fig. 3. Command Widget.

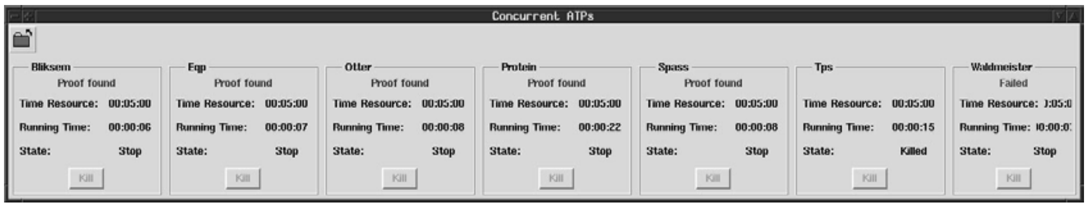
by their type: elimination rules, introduction rules, structural rules, etc. The menu for tactics is divided into sub-menus according to the theories the tactics belong to. These sub-menus can be further classified according to the categories specified within these theories. This supports the user in navigating and finding appropriate inferences. An inference can be listed in several subtopics

Inferences are applied by executing the attached commands. In general, it is necessary to provide some arguments for the application of an inference, which can be specified inside a generic command window. The command window adjusts itself automatically to the number of arguments and provides some help to find the requested parameters (cf. Fig. 3 for an example). The user specifies the arguments either by manually entering them or by a mouse-click on the appropriate node.

Planner. ΩMEGA's proof planner searches in the space of partial proof plans, where methods are the plan operators. The planner transforms a partial $\mathcal{PDS}$ by method application and by recursive method expansion until it obtains a complete $\mathcal{PDS}$, i.e. a $\mathcal{PDS}$ which represents (can be expanded into) an ND proof.

The commands for ΩMEGA's planner are grouped into $\mathcal{L}\Omega\mathcal{UI}$'s planner menu. The interface displays the growing partial plan as an abstraction of the $\mathcal{PDS}$.

External Systems. ΩMEGA employs several ATPs, constraint solvers, and CASs as external reasoners to be applied to specific proof problems. Automated theorem provers that are currently available to ΩMEGA are the first-order systems OTTER, SPASS, PROTEIN, BLIKSEM, SATCHMO, WALDMEISTER, and EQP, (cf. [BCF97, HuF96, FrK99]) and the higher-order theorem provers TPS (cf. [BeS98b] and LEO [BeK98]. The computer algebra systems include the experimental system μCAS (cf. [KKS98]) as well as the full-blown systems GAP, MAPLE, and MAGMA.


Concurrent ATPs

Bliksem	Eqp	Otter	Protein	Spass	Tps	Waldmeister
Proof found	Proof found	Proof found	Proof found	Proof found		Failed
Time Resource: 00:05:00	Time Resource: 00:05:00	Time Resource: 00:05:00	Time Resource: 00:05:00	Time Resource: 00:05:00	Time Resource: 00:05:00	Time Resource:):05:0
Running Time: 00:00:06	Running Time: 00:00:07	Running Time: 00:00:08	Running Time: 00:00:22	Running Time: 00:00:08	Running Time: 00:00:15	Running Time: I0:00:0
State: Stop	State: Stop	State: Stop	State: Stop	State: Stop	State: Killed	State: Stop

Fig. 4. Concurrent ATPs.

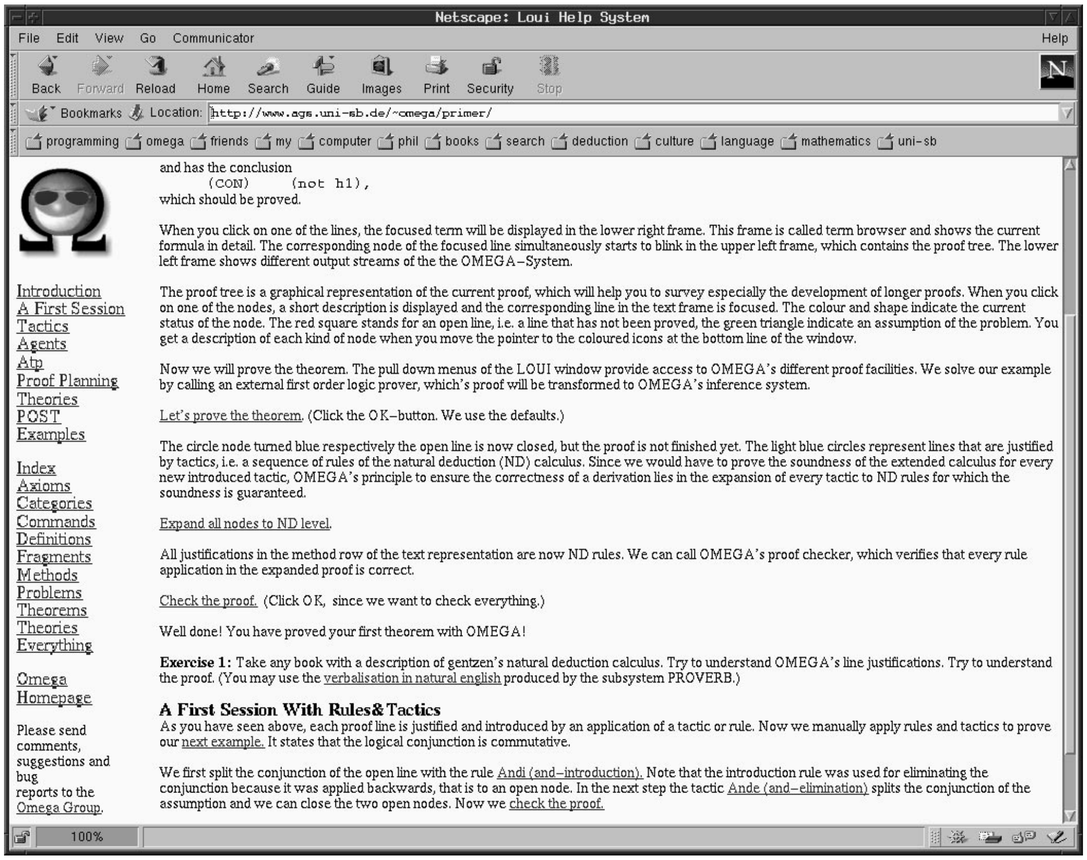
An interesting aspect of ΩMEGA is its ability to employ several ATPs concurrently. The graphical user interface supports the control of parallel ATPs by providing a special widget as displayed in Fig. 4, which helps the user to monitor the activities of every single running ATP. Messages report the status of an ATP as either not available, still running, or whether the prover has found a proof or failed to do so. The window enables the user to interactively tailor the time resources given to single ATPs or to kill running processes entirely.

Command Suggestion Mechanism. Another feature of ΩMEGA that can only be fully exploited with a graphical user interface is its elaborate mechanism to guide the user when interactively proving a theorem. It suggests commands, applicable in the current proof state – more precisely commands that invoke some ND-rules or tactics – together with sets of suitable instantiations of the command arguments. It is based on two layers of autonomous, concurrent agents which steadily work in the background of the ΩMEGA system and dynamically update their computational behaviour with respect to the state of the proof and/or specific user queries of the suggestion mechanism. By exchanging information via blackboards the agents cooperatively accumulate useful command suggestions which are then heuristically sorted and presented to the user.

These suggestions are displayed in a separate window (see the right side of Fig. 1). The entries of this window are constantly updated by the suggestion mechanism, which is based on the computation of the society of agents. The command that is most likely to be useful in the current proof state is always in the first position. However, the user can choose any of the proposed commands. As long as the command is not yet executed (by confirming the argument settings in the corresponding command widget), the suggestions are still updated, giving the user the opportunity to rethink his decision.

The suggestion mechanism always proposes a command together with several possible sets of instantiations for the command's arguments. Such a set is a meaningful combination of parameters, e.g. proof lines, terms, etc., the command is applicable to. The sets are heuristically sorted and the *best* one is immediately proposed to the user by providing it as default arguments to the command. However, all other computed sets of argument instantiations can still be displayed and chosen within the respective command widget. One can browse the suggestions or ask for a single argument (cf. the arrow buttons on the lower left and next to the arguments in Fig. 3, respectively). Furthermore, the user can specify one or several of the arguments as fixed and call the suggestion mechanism again by clicking on the recompute button (cf. Fig. 3) to get possible suggestions for the remaining parameters.

Online Help. The online documentation of ΩMEGA contains descriptions of commands, planning methods, and the content of the theories. It is based on HTML and can be viewed with any standard web browser. Besides the usual references to related help topics, the hyperlink mechanism is extended to allow interaction with $\mathcal{L}\Omega\mathcal{UI}$, i.e. the execution of commands and call of menus by clicking on links within a help page. This yields a nice tool that is used inter alia to introduce new users to the system (see Fig. 5). Furthermore the documentation of theorems and problems contains commands to import them into a current problem or to execute a proof script respectively.


Netscape: Loui Help System
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Images Print Security Stop
Bookmarks Location: http://www.ags.uni-sb.de/~omega/primer/
programming omega friends my computer phil books search deduction culture language mathematics uni-sb
Introduction
A First Session
Tactics
Agents
Atp
Proof Planning
Theories
POST
Examples
Index
Axioms
Categories
Commands
Definitions
Fragments
Methods
Problems
Theorems
Theories
Everything
Omega
Homepage
100%

Please send comments, suggestions and bug reports to the Omega Group.

and has the conclusion

```
      (CON)     (not h1),
```

which should be proved.

When you click on one of the lines, the focused term will be displayed in the lower right frame. This frame is called term browser and shows the current formula in detail. The corresponding node of the focused line simultaneously starts to blink in the upper left frame, which contains the proof tree. The lower left frame shows different output streams of the the OMEGA–System.

The proof tree is a graphical representation of the current proof, which will help you to survey especially the development of longer proofs. When you click on one of the nodes, a short description is displayed and the corresponding line in the text frame is focused. The colour and shape indicate the current status of the node. The red square stands for an open line, i.e. a line that has not been proved, the green triangle indicate an assumption of the problem. You get a description of each kind of node when you move the pointer to the coloured icons at the bottom line of the window.

Now we will prove the theorem. The pull down menus of the LOUI window provide access to OMEGA's different proof facilities. We solve our example by calling an external first order logic prover, which's proof will be transformed to OMEGA's inference system.

Let's prove the theorem. (Click the OK–button. We use the defaults.)

The circle node turned blue respectively the open line is now closed, but the proof is not finished yet. The light blue circles represent lines that are justified by tactics, i.e. a sequence of rules of the natural deduction (ND) calculus. Since we would have to prove the soundness of the extended calculus for every new introduced tactic, OMEGA's principle to ensure the correctness of a derivation lies in the expansion of every tactic to ND rules for which the soundness is guaranteed.

Expand all nodes to ND level.

All justifications in the method row of the text representation are now ND rules. We can call OMEGA's proof checker, which verifies that every rule application in the expanded proof is correct.

Check the proof. (Click OK, since we want to check everything.)

Well done! You have proved your first theorem with OMEGA!

Exercise 1: Take any book with a description of gentzen's natural deduction calculus. Try to understand OMEGA's line justifications. Try to understand the proof. (You may use the verbalisation in natural english produced by the subsystem PROVERB.)

A First Session With Rules&Tactics

As you have seen above, each proof line is justified and introduced by an application of a tactic or rule. Now we manually apply rules and tactics to prove our next example. It states that the logical conjunction is commutative.

We first split the conjunction of the open line with the rule Andi (and–introduction). Note that the introduction rule was used for eliminating the conjunction because it was applied backwards, that is to an open node. In the next step the tactic Ande (and–elimination) splits the conjunction of the assumption and we can close the two open nodes. Now we check the proof.

Fig. 5. The online help system.

5. The Agent Architecture

A mathematical assistant system calls for an open and distributed system architecture. In such an architecture, the developer of a deduction system or a respective tool upgrades it to a so-called *mathematical service* by wrapping it into an agent-oriented parcel, with an interface to a common *mathematical software bus* [Hom96, HoC96].

The functionality of the software bus is realized in the MathWeb system (see [FrK99, FHJ99] for details) by the trader model shown in Fig. 6, which limits the central administration of the bus to a so-called *trading point* that provides routing and authentification information to the mathematical services. We will describe this model in more detail below.

We have implemented and experimented with our MathWeb system, where the integrated theorem proving systems and tools can be distributed worldwide and interact over the Internet. They can be dynamically added to and subtracted from the coordinated reasoning repertoire of the complete system. For this, we have embedded all of ΩMEGA's external reasoners presented in the previous section. Now, they jointly handle an average load of tens of thousands of inference problems a day.

$\mathcal{L}\Omega\mathcal{UI}$ is now just one of the cooperating mathematical services, but it is special in the sense that it is the only part directly visible to and interacting with the user. In this way it has to meet special requirements: $\mathcal{L}\Omega\mathcal{UI}$ must be able to display the mathematical services that are currently active, it must be able to suggest external services that may be appropriate for the current task, and it

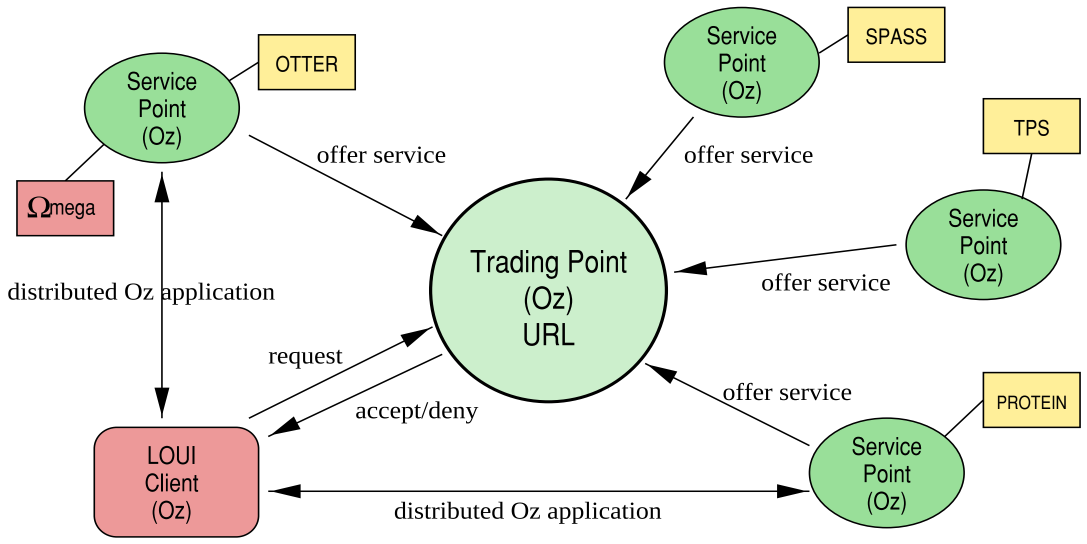


Fig. 6. The architecture for distributed mathematical services.

must be able to control external systems of which it might not always be able to get complete information.

Locating the services that are currently available is achieved by providing $\mathcal{L}\Omega\mathcal{UI}$ with a list of services that might be available, and it then tries to contact each of these services during initialization and tests whether a connection with the service is possible. Apart from this, $\mathcal{L}\Omega\mathcal{UI}$'s own representation of the service's capabilities is currently limited to a general classification of the service, for instance whether it is a theorem prover or a computer algebra system, and some information about command syntax and flag settings that are important for controlling the service needs extension by a list of more specific qualities and weaknesses of the service.

Given its limited knowledge about collaborating services, $\mathcal{L}\Omega\mathcal{UI}$'s answer to the problem of finding the right one is pragmatic: it simply starts all problem-related services on the mathematical bus. The user can eliminate those reasoners that she thinks might not be suitable to solve the problem. $\mathcal{L}\Omega\mathcal{UI}$ will call selected services in parallel in order to maximize the likelihood of success and minimize the waiting time of the user. The use of several reasoners for the same task has the additional advantage of cross-validating results if needed. In any case, the proofs found by these systems have to be transformed into the internal proof format of the ΩMEGA system; this proof transformation process itself runs in parallel to the ongoing user interaction.

The Maintenance Advantage. The agent architecture that separates ΩMEGA's logical kernel from its graphical user interface has increased both its efficiency and maintainability.

It is quite common in local computer networks that users have relatively low-speed machines on their desktop, whereas some high-speed servers that are accessible for everyone operate in the background. Running the user interface on the local machine uses the local resources that are sufficient for this task, while the more powerful servers can be exploited for the really complex task of actually searching for a proof.

The maintenance advantage applies to both the user and the developer. ΩMEGA is a rather large system (roughly 17 MB of COMMON LISP (CLOS) code for the main body of the current version), comprising numerous associated modules (such as the integrated automated theorem provers and computer algebra systems) written in various programming languages. It is a difficult task to install the complete system, in particular, successful installation depends on the presence of (proprietary) compilers or interpreters for the respective programming languages.

In our architecture the user only installs the $\mathcal{L\Omega UI}$ client, which connects to the main system and exchanges data with it via the Internet. Thus the user interacts with the client, which can be physically anywhere in the world, while the ΩMEGA kernel is still on our server (here in Saarbrücken, where it is maintained and developed). Since $\mathcal{L\Omega UI}$ is realized via the distributed programming system environment Mozart/Oz [Smo95],[4] which is freely available for various platforms including UNIX and the MS-Windows family of operating systems, this keeps the software and hardware requirements of the user moderate. The installation of the client is further simplified by running $\mathcal{L\Omega UI}$ as a Netscape applet, i.e. $\mathcal{L\Omega UI}$ is automatically downloaded via the Internet. Thus we are able to provide current versions of ΩMEGA and $\mathcal{L\Omega UI}$ without the need for re-installation at the user's site.

To reduce the bandwidth needed for communication, ΩMEGA implements an incremental approach based on SMALLTALK's MVC triad[5], which only transmits the parts of the $\mathcal{PDS}$ that are changed by a user action. This not only improves response time for a low-bandwidth Internet connection but also focuses the user's attention to the effects of an action.

Since the presentation of the proof tree is defined by a context-free grammar, it is rather easy to connect $\mathcal{L\Omega UI}$ to provers other than ΩMEGA; we have experimented with LEO [BeK98] and INKA [HuS96] and λ-CLAM [RSG98]. In this sense $\mathcal{L\Omega UI}$ can be seen as a generic proof viewer.

Distributing ΩMEGA. Up to this point, we have considered a client-server network with one server that is dedicated to ΩMEGA itself and several clients that use this server. In reality, a ΩMEGA network may consist of several servers that can be accessed via a gateway service. The gateway daemon runs on one machine that provides the ΩMEGA service. It can start the actual ΩMEGA process and its associated modules on any of the servers, depending on their current work load. In this way, we are able to employ the whole computational power of a local area network with a background of several larger servers.

6. Related Work

User interfaces are the subject of an important discipline in computer science with its own conferences, workshops and research groups. Many industrial applications spend substantial effort just on the interface with up to eighty percent of the systems source code being developed for a friendly interface. The techniques and methods of this discipline slowly but surely find their way even to rather theoretically oriented fields such as automated theorem proving, where the importance is increasingly well recognized.

Most interfaces of ATP systems provide graphical illustrations of proof structures and their elements, and facilities to set up commands in the proof environ-

[4] `http://www.mozart-oz.org/`

[5] See for instance `http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html` for an overview.

ment. The semantics of proof steps are often expressed by graphical objects and annotations. Examples for this sort of visualization are binary decision diagrams for first-order deduction systems [PoS95], which have special display facilities for the relation between quantified formulae and their instantiation, and natural deduction displays of sequent proofs [Bor97] where the scoping structure of the proof is visualized by adjacent or by nested boxes enclosing segments of proof lines. Another presentation technique displays proof steps in an appropriately formatted and interactive way. The THEOREMA system [BJK97] can present a proof in natural language by employing fixed natural language schemata. Details that are temporarily hidden can be exposed by clicking on the corresponding root of the proof line.

A verbalization component on top of Nuprl uses a natural language generator and presents proofs at the level of tactics [HMB99].

CTCOQ [BKT94] is a rather elaborate presentation system which distributes the proof information about a proof over three sections of a multi-paned window: a *Command* window records the script of commands sent to the proof engine, a *State* window contains the current goals to be proved, and a *Theorems* window contains the results of queries into the proof engine's theorem database.

Other approaches put particular emphasis on visualization by making the tree format of the proof structure explicit in the display. The user interface of the SEAMLESS system [EuM97] provides display facilities for a proof graph at different levels of abstraction in a framed window: a variety of lay-out operations includes zooming and reuse of lemmata.

The user interface of INKA [HuS96] allows for the display of induction proof sketches at varying levels of detail. Its features include status information, typically expressed by different colouring, context-sensitive menus of possible user actions, and proof by pointing.

The proof verification systems VSE [HLS96] has a very elaborate user interface that enables the proof engineer to verify industrial software by visualizing relations between underlying theories (specifications).

The ILF system [Dah98] uses an interface to display proofs in natural language and in a tree like structure, where the logical content of nodes is displayed separately. Furthermore, queries can be sent to the MIZAR library and several first-order automatic theorem provers running in parallel under control of the interface.

ΩMEGA in some sense combines features of SEAMLESS, CTCOQ, and ILF. Its graphical display is similar to that of SEAMLESS, but the set of node categories and their display is fixed to the particular proof environment. However, $\mathcal{L}\Omega\mathcal{UI}$'s tree visualization can easily be adapted to a different set of node categories and display options. Its display of status information is similar to that of CtCoq, but the database window is handled differently. The concurrent handling of external reasoners is related to ILF, but since ΩMEGA's logic is higher-order, a larger variety of automatic systems has been integrated. The handling of co-references and the combination of tree-like and linear display together with the hyper-link mechanism to visualize references between both are unique to $\mathcal{L}\Omega\mathcal{UI}$.

7. Conclusions

$\mathcal{L}\Omega\mathcal{UI}$ represents an agent-based, distributed approach to user interfaces for theorem provers. It provides advanced communication facilities via an adaptable proof tree visualization technique and through various selective proof object display methods which enable the user to better understand the proof and to guide the proof search.

Even though $\mathcal{L}\Omega\mathcal{UI}$ was originally developed for the ΩMEGA system it is not restricted to it in principle. We have also used it as an independent interface to various other deduction systems. However much more work has to be invested to generalize it to a universal user interface for theorem provers.

References

[BCF97] Benzmüller, C., Cheikhrouhou, L., Fehrer, D., Fiedler, A., Huang, X., Kerber, M., Kohlhase, M., Konrad, K., Melis, E., Meier, A., Schaarschmidt, W., Siekmann, J. and Sorge, V.: ΩMEGA: Towards a mathematical assistant. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, number1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

[BJK97] Buchberger, B., Jebelean, T., Kriftner, F., Marin, M., Tomuta, E. and Vasaru, D.: An Overview of the Theorema Project. In *ISSAC'97*, Hawaii, 1997.

[BeK98] Benzmüller, C. and Kohlhase, M.: LEO, a Higher Order Theorem Prover. In Kirchner and Kirchner [KiK98], pages 139–144.

[BKT94] Bertot, Y., Kahn, G. and Therry, L.: Proof by Pointing. *Theoretical Aspects of Computer Software*, 789:141–160, 1994.

[Bac98] Backhouse, R. C.: editor. *Proceedings of the Workshop on User Interfaces for Theorem Provers*, number98/08 in Computing Science Reports, Eindhoven, the Netherlands, 1998. Eindhoven University.

[Bor97] Bornat, R.: Natural Deduction Displays of Sequent Proofs: Experience with the Jape Calculator. In *First International Workshop on Proof Transformation and Presentation*, Dagstuhl Castle, 1997.

[BeS98a] Benzmüller, C. and Sorge, V.: A Focusing Technique for Guiding Interactive Proofs. Submitted to the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications, 1998.

[BeS98b] Benzmüller, C. and Sorge, V.: Integrating TPS with ΩMEGA. In Jim Grundy and Malcolm Newey, editors, *Theorem Proving in Higher Order Logics: Emerging Trends*, Technical Report 98-08, Department of Computer Science and Computer Science Lab, pages 1–19, Canberra, Australia, October 1998. The Australian National University. available from http://cs.anu.edu.au/techreports/recent.html.

[ChS98] Cheikhrouhou, L. and Siekmann, J.: Planning diagonalization proofs. In *Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, LNAI, Sozopol, Bulgaria, 1998.

[Che98] Cheikhrouhou, L.: A multi-modi Proof Planner. In J. Dix and S. Hölldobler, editors, *Inference Mechanisms in Knowledge-Based Systems: Theory and Application (Proceedings of WS at KI'98)*, Fachberichte INFORMATIK 19/98, pages 91–102, Koblenz, Germany, September 1998. University of Koblenz-Landau.

[Dah98] Dahn, I.: Using ILF as a User Interface for Many Theorem Provers. In Backhouse [Bac98], pages 75–86.

[DaH93] Dalianis, H. and Hovy, E. H.: Aggregation in Natural Language Generation. In M. Zock, G. Adorni and G. Ferrari, editors, *Proceedings of the 4th European Workshop on Natural Language Generation*, pages 67–73, 1993.

[Eas98] Eastaughffe, K.: Support for Interactive Theorem Proving: Some Design Principles and Their Application. In Backhouse [Bac98], pages 96–103.

[EuM97] Eusterbrock, J. and Michalis, N.: A World-Wide Web Interface for the Visualization of Constructive Proofs at Different Abstraction Layers. In *First International Workshop on Proof Transformation and Presentation*, Dagstuhl Castle, 1997.

[FHJ99] Franke, A., Hess, S. M., Jung, C. G., Kohlhase, M. and Sorge, V.: Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 5:3, pp. 156–187, 1999.

[FrK99] Franke, A. and Kohlhase, M.: System description: MATHWEB, an agent-based communication layer for distributed automated theorem proving. to appear in CADE'99, 1999.

[HoC96] Homann, K. and Calmet, J.: Structures for Symbolic Mathematical Reasoning and Computation. In J. Calmet and C. Limogelli, editors, *Design and Implementation of Symbolic Computation Systems, DISCO'96*, number1128 in LNCS, pages 216–227, Karlsruhe, Germany, 1996. Springer Verlag.

[HuF96] Huang, X. and Fiedler, A.: Presenting Machine-Found Proofs. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, number1104 in LNAI, pages 221–225, New Brunswick, NJ, USA, 1996. Springer Verlag.

[HuF97] Huang, X. and Fiedler, A.: Proof Verbalization as an Application of NLG. In M. E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, Nagoya, Japan, 1997. Morgan Kaufmann.

[HLS96] Hutter, D., Langenstein, B., Sengler, C., Siekmann, J. H., Stephan, W. and Wolpers, A.: Verification support environment. *High Integrity Systems*, 1(6), 1996.

[HMB99] Holland-Minkley, A. M., Barzilay, R. and Constable, R. L.: Verbalization of high-level formal proofs. In *National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[Hom96] Homann, K.: *Symbolisches Lösen mathematischer Probleme durch Kooperation algorithmischer und logischer Systeme*. PhD thesis, Unversität Karlsruhe, 1996.

[HuS96] Hutter, D. and Sengler, C.: A Graphical User Interface for an Inductive Theorem Prover. In *International Workshop on User Interface Design for Theorem Proving Systems*, 1996.

[KiF95] Kilger, A. and Finkler, W.: TAG-Based Incremental Generation. *Computational Linguistics*, 1995.

[KiK98] Kirchner, C. and Kirchner, H.: editors. *Proceedings of the 15th Conference on Automated Deduction*, number1421 in LNAI, Lindau, Germany, 1998. Springer Verlag.

[KKS98] Kerber, M., Kohlhase, M. and Sorge, V.: Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 1998. Special Issue on the Integration of Computer Algebra and Automated Deduction; forthcoming.

[McC94] McCune, W. W.: Otter 3.0 Reference Manual and Guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.

[Met92] Meteer, M. W.: *Expressibility and the Problem of Efficient Text Planning*. Pinter Publishers, London, 1992.

[Mel98] Melis, E.: AI-techniques in proof planning. In *European Conference on Artificial Intelligence (ECAI-98)*, pages 494–498, Brighton, 1998. Kluwer.

[MeL99] Melis, E. and Leron, U.: A proof presentation suitable for teaching proofs. In *9th International Conference on Artificial Intelligence in Education, AI-ED'99*, Le Mans, 1999. IOS Press.

[PoS95] Posegga, J. and Schneider, K.: Interactive First-Order Deduction with BDDs. In *International Workshop on User Interface Design for Theorem Proving Systems*, Glasgow, 1995.

[Rei85] Reichman, R.: *Getting Computers to Talk Like You and Me*. MIT Press, 1985.

[Rei94] Reiter, E.: Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, Kennebunkport, Maine, USA, 1994.

[RSG98] Richardson, J., Smaill, A. and Green, I.: Proof Planning in Higher-Order Logic with $\lambda Clam$. In Kirchner and Kirchner [KiK98], pages 139–144.

[Shn92] Shneiderman, B.: *Designing the User Interface*, Volume 2. Addison-Wesley, 1992.

[Smo95] Smolka, G.: The Oz Programming Model. In J. v. Leeuwen, editor, *Computer Science Today*, Volume 1000 of *LNCS*, pages 324–343. Springer-Verlag, 1995.

Received November 1998
Accepted in revised form June 1999