$\mathcal{L}\Omega\mathcal{UI}$: A Distributed Graphical User Interface for the Interactive Proof System ΩMEGA

Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou,
Detlef Fehrer, Armin Fiedler, Helmut Horacek, Michael Kohlhase,
Karsten Konrad, Andreas Meier, Erica Melis, Volker Sorge,
FB Informatik, Universität des Saarlandes, Germany
`http://www.ags.uni-sb.de`

Abstract

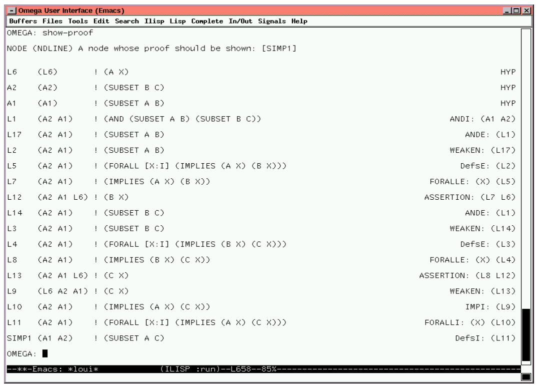
Most interactive proof development environments are insufficient to handle the complexity of the information to be conveyed to the user and to support his orientation in large-scale proofs. In this paper we present a distributed client-server extension of the ΩMEGA proof development system, focusing on the $\mathcal{L}\Omega\mathcal{UI}$ (**L**ovely ΩMEGA **U**ser **I**nterface) client. This graphical user interface provides advanced communication facilities through an adaptable proof tree visualization and through various selective proof object display methods. Some of $\mathcal{L}\Omega\mathcal{UI}$'s main features are the graphical display of co-references in proof graphs, a selective term browser, and support for dynamically adding knowledge to partial proofs – all based upon and implemented in a client-server architecture.

1 Introduction

One (of several) reasons, why current deduction systems have not found a wider acceptance in mathematical practice is that they are too inconvenient to use. The ΩMEGA system [BCF$^+$97] – an interactive, plan-based deduction system with the ultimate goal of supporting theorem proving in main-stream mathematics and mathematics education must address this, in order to reach its goal. In order to provide a conceptually structured, understandable and easily usable front-end, the interface $\mathcal{L}\Omega\mathcal{UI}$ of the ΩMEGA system is designed with respect to the following requirements:

- 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 a proof tree, as a linearized proof, or in verbalization mode, etc.).
- The system should minimize the necessary interaction by suggesting commands and parameters to the user in each proof step. Optimally, the system should be able to do all straight-forward steps autonomously.
- The interface should work reasonably fast, and its installation in other environments should be possible with minimal effort and storage requirement.

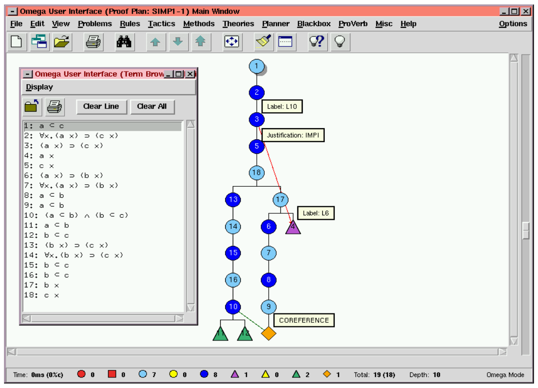
These issue are elaborated in detail in the following three sections. We will only discuss the ΩMEGA proof system (the current system consists of a proof planner and an integrated collection of tools for formulating problems, proving subproblems, and proof presentation) where it becomes necessary to understand the interface issues.


Omega User Interface (Emacs)
Buffers Files Tools Edit Search Ilisp Lisp Complete In/Out Signals Help

```
OMEGA: show-proof

NODE (NDLINE) A node whose proof should be shown: [SIMP1]

L6    (L6)        ! (A X)                                                   HYP
A2    (A2)        ! (SUBSET B C)                                            HYP
A1    (A1)        ! (SUBSET A B)                                            HYP
L1    (A2 A1)     ! (AND (SUBSET A B) (SUBSET B C))                ANDI: (A1 A2)
L17   (A2 A1)     ! (SUBSET A B)                                    ANDE: (L1)
L2    (A2 A1)     ! (SUBSET A B)                                 WEAKEN: (L17)
L5    (A2 A1)     ! (FORALL [X:I] (IMPLIES (A X) (B X)))           DefsE: (L2)
L7    (A2 A1)     ! (IMPLIES (A X) (B X))                    FORALLE: (X) (L5)
L12   (A2 A1 L6) ! (B X)                                   ASSERTION: (L7 L6)
L14   (A2 A1)     ! (SUBSET B C)                                    ANDE: (L1)
L3    (A2 A1)     ! (SUBSET B C)                                 WEAKEN: (L14)
L4    (A2 A1)     ! (FORALL [X:I] (IMPLIES (B X) (C X)))           DefsE: (L3)
L8    (A2 A1)     ! (IMPLIES (B X) (C X))                    FORALLE: (X) (L4)
L13   (A2 A1 L6) ! (C X)                                  ASSERTION: (L8 L12)
L9    (L6 A2 A1) ! (C X)                                         WEAKEN: (L13)
L10   (A2 A1)     ! (IMPLIES (A X) (C X))                           IMPI: (L9)
L11   (A2 A1)     ! (FORALL [X:I] (IMPLIES (A X) (C X)))    FORALLI: (X) (L10)
SIMP1 (A1 A2)     ! (SUBSET A C)                                   DefsI: (L11)
OMEGA: 
--**-Emacs: *loui*          (ILISP :run)--L658--85%----------------------------
```


Omega User Interface (Proof Plan: SIMP1-1) Main Window
File Edit View Problems Rules Tactics Methods Theories Planner Blackbox ProVerb Misc Help Options
Omega User Interface (Term Brow
Display
Clear Line
Clear All
1
2
Label: L10
3
Justification: IMPI
5
18
13
17
Label: L6
14
6
4
15
7
16
8
10
9
COREFERENCE
11
12

```
1: a ⊂ c
2: ∀x.(a x) ⊃ (c x)
3: (a x) ⊃ (c x)
4: a x
5: c x
6: (a x) ⊃ (b x)
7: ∀x.(a x) ⊃ (b x)
8: a ⊂ b
9: a ⊂ b
10: (a ⊂ b) ∧ (b ⊂ c)
11: a ⊂ b
12: b ⊂ c
13: (b x) ⊃ (c x)
14: ∀x.(b x) ⊃ (c x)
15: b ⊂ c
16: b ⊂ c
17: b x
18: c x
```

Time: 0ms (0%c) 🔴 0 🟥 0 ● 7 🟡 0 🔵 8 △ 1 △ 0 △ 2 ◆ 1 Total: 19 (18) Depth: 10 Omega Mode

Figure 1: A linearized proof and its graphical tree-representation in $\mathcal{L}\Omega\mathcal{UI}$

2 Multi-modal Views: Proof Tree Visualization and Proof Content Display

The ΩMEGA system provides different techniques to analyze partial or complete proofs. As in traditional theorem proving systems, $\mathcal{L}\Omega\mathcal{UI}$ can present a proof in a linearized form, in our case as a higher-order variant of Gentzen's Natural Deduction (ND) calculus (see figure 1). For long proofs, such a presentation lacks transparency and structure. Therefore $\mathcal{L}\Omega\mathcal{UI}$ offers two additional ways of representing proofs: as a tree that models the logical dependencies between the different proof lines and as a text in natural language, as it would appear in a mathematical textbook. Before we go into details let us look at an example:

Example 2.1 ***(Proof Representations in*** $\mathcal{L}\Omega\mathcal{UI}$***).*** The left window in figure 1 shows the linearized ND format of a simple proof of the transitivity of the subset relation, while the right window shows $\mathcal{L}\Omega\mathcal{UI}$'s main window with a tree representation of the proof. Below, we have a natural-language representation of the same proof that has been automatically generated by the ΩMEGA system.

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

Proof:
Let $x \in a$. That implies that we have $x \in b$. That leads to $x \in c$. We have $a \subset c$ since $\forall x.\ x \in a \Rightarrow x \in c$. □

2.1 Hierarchical Plan Data Structure

The entire process of theorem proving in ΩMEGA can be viewed as an interleaving process of proof planning, plan execution, and verification that is centered around the so-called *Proof Plan Data Structure* ($\mathcal{PDS}$).

The hierarchical data structure represents a (partial) proof at different levels of abstraction (called *proof plans*). It is represented as a directed acyclic graph, where the nodes are justified by methods. Conceptually, each justification represents a proof plan (the *expansion* of the justification) at a lower level of abstraction that is computed when the method is expanded. A proof plan can be recursively expanded, until a fully explicit proof on the calculus level (ND) has been reached. In ΩMEGA, we keep the original proof plan in an expansion hierarchy. 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.

Once a proof plan is completed, its justifications can successively be expanded to verify the well-formedness of the ensuing $\mathcal{PDS}$. When the expansion process is completed, the establishment of correctness of the ND proof relies solely on the correctness of the verifier and the calculus. This approach also provides a basis for a controlled integration of external reasoning components – such as an automated theorem prover or a computer algebra system – if each reasoner's results can (on demand) be transformed into a sub-$\mathcal{PDS}$.

A $\mathcal{PDS}$ can be constructed by automated or mixed-initiative planning, or by pure user interaction. In particular, new pieces of the $\mathcal{PDS}$ can be added by directly calling tactics, by inserting facts from a data base, or by calling some external reasoner. Automated proof planning is only adequate for problem classes for which method and control knowledge have already been established.

2.2 Visualization – Proofs as Trees

In the main display window of $\mathcal{L}\Omega\mathcal{UI}$, the structure of proofs is shown in a pure tree format, independently of the logical terms associated with the nodes (see the central part in Figure 1). Since logical proofs are in general acyclic directed graphs and not trees, $\mathcal{L}\Omega\mathcal{UI}$ 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 – that points to the root of the displayed subproof. Thus the resulting structure is a proper tree, which is displayed in such a way that node categories are expressed by color and shape (see the front panel of the window in the right part of figure 1):

Terminal nodes are represented by triangles, with assumptions, assertions, and hypotheses distinguished according to their color (green, yellow, and violet).

Intermediate nodes are represented as circles, with ground, expanded, unexpanded, and open nodes distinguished according to their color (dark blue, bright blue, yellow, and red).

Untested nodes are represented by red squares. A node is considered untested in case ΩMEGA assumes an external reasoner to be able to solve the associated sub-problem but the assumption is not yet verified.

Co-reference nodes, which may or may not be terminal nodes, are represented by diamonds and uniquely colored in orange.

The categories of intermediate nodes need some explanation. While open nodes are subject to further derivations, the other nodes are distinguished by their respective level of abstraction in the $\mathcal{PDS}$. Ground nodes are at the ND level, while all others are on higher levels of abstraction; Expanded nodes are nodes, where the expansion to the natural deduction level is known, but not displayed. The user has the following possibilities to manipulate the appearance of the proof 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.

2.3 Term and Proof Content Display

The design decision to separate the tree structure from the terms associated with individual nodes enables the display of large trees without crowds of annotations. The connection between the tree structure and the associated content can be selectively re-established by the user. One possibility to achieve this is the introduction of annotations by clicking at a node, so that a yellow box enclosing a label and a justification appears besides that node (four such boxes appear in Figure 1). Another possibility is to apply the *term browser* (see the smaller window beside of the proof tree in Figure 1): by double-clicking at some node the associated term is displayed in the term browser. Nodes whose terms appear in the term browser are numbered dynamically in the displayed proof tree. Pointing to either a node or a term leads to both objects being highlighted in their respective windows. Co-references are not handled by the term browser. Instead, pointing to a co-reference node leads to the temporary appearance of a line between the co-reference node and the node it co-refers to.

2.4 Proofs in Natural Language: Proverb

Ωmega uses an extension of the Proverb system [HF97] developed in our group that presents proofs and proof plans in natural language. In order to produce coherent texts that resemble those found in mathematical textbooks, Proverb employs state-of-the-art techniques of natural language processing and generation.

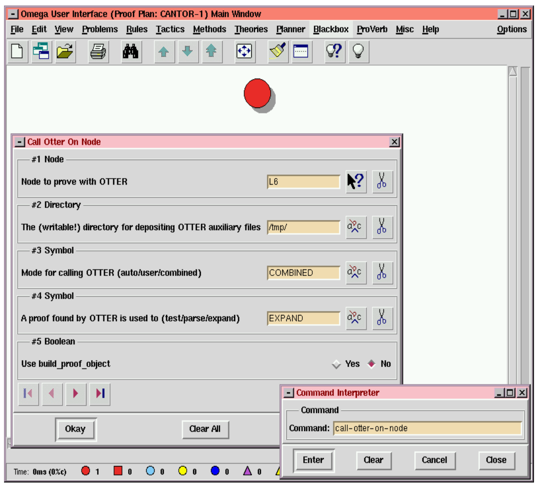
Due to the possibly hierarchical nature of $\mathcal{PDS}$ proofs, these can be verbalized at more than one level of abstraction, which can be selected by the user. Since a user will normally want to vary the level of abstraction in the course of a proof, the current verbalization facility will be extended to one that explains proofs to users guided by their feedback.

3 Controlling Ωmega

Ωmega's main functionality – especially including those commands and facilities important for interactive proof development – is available via the structured menu bar in $\mathcal{L}\Omega\mathcal{UI}$'s main window. Its entries reflect the conceptually different facilities of the Ωmega-system. For instance, there are a menu entities *Black-box* and *Planner* providing all useful commands of these conceptual categories to the user.

For non-experts and especially for novices, a graphical user interface has many advantages over a purely command-shell based user interface, as it provides a steady overview on the – mostly unknown – system commands to the user and thus relieves him from searching for appropriate commands in an interactive shell. Experts, who are familiar with nearly all of the commands, may prefer the interaction via a command-shell. Therefore, $\mathcal{L}\Omega\mathcal{UI}$ also provides a command shell for expert users (see the bottom-right part of the figure).

One important feature of $\mathcal{L}\Omega\mathcal{UI}$ is its dynamic and generic menu extension, i.e. $\mathcal{L}\Omega\mathcal{UI}$ selectively offers commands to the user depending on the current system state. This is in contrast to most systems which always present all commands even if some of them do not make sense within the current state. For instance, when working on a problem within a given theory, only those commands will be offered which belong to this theory (or it's parent theories) or which are defined to be always applicable. This generic approach eases the integration of new commands as they can be either integrated fully automatically by connecting them with a certain theory or by just adding the command-name to one of the non-dynamic menu entities.


Omega User Interface (Proof Plan: CANTOR-1) Main Window
File Edit View Problems Rules Tactics Methods Theories Planner Blackbox ProVerb Misc Help Options
Okay
Clear All
Enter
Clear
Cancel
Close
Time: 0ms (0%c) ● 1 ■ 0 ○ 0 ○ 0 ● 0 △ 0

Call Otter On Node

#1 Node
Node to prove with OTTER: L6

#2 Directory
The (writable!) directory for depositing OTTER auxiliary files: /tmp/

#3 Symbol
Mode for calling OTTER (auto/user/combined): COMBINED

#4 Symbol
A proof found by OTTER is used to (test/parse/expand): EXPAND

#5 Boolean
Use build_proof_object: Yes [✓ No]

Command Interpreter

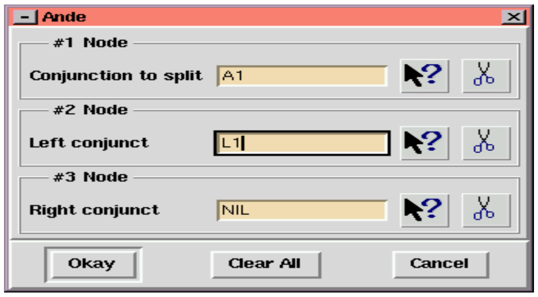
Command
Command: call-otter-on-node

In the following subsections, we illustrate the connection of $\mathcal{L}\Omega\mathcal{UI}$ to major parts of Ωmega.

Theories In Ωmega, mathematical knowledge is structured with respect to mathematical domains and is therefore organized in a hierarchy of theories. Theories represent signature extensions, 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 problem has been proven by constructing and verifying a proof it can be stored into the theory where it was proven.

Rules, Tactics and Methods The hierarchic organization of theories and their incremental importation does not only affect their availability for a proof but also $\mathcal{L}\Omega\mathcal{UI}$'s menu structure. Since theories contain rules, tactics and planning methods, these are also incrementally loaded. To each inference method there is an attached command which is not statically contained in the interface but is dynamically appended to the menu structure. Commands for inference methods are inserted into the respective menu topic for rules, tactics, and methods. Within these topics, the commands are ordered in additional sub-menus. Since rules are always defined in ΩMEGA's base theory, they are just sorted by their type: elimination rules, introduction rules, structural rules, etc. The menus for both, methods and tactics, are divided into sub-menus according to the theories the inference methods belong to. These sub-menus can be further divided by categories specified within these theories. Moreover, each inference method can be listed in several subtopics in the menus.


Ande

#1 Node

Conjunction to split: A1

#2 Node

Left conjunct: L1

#3 Node

Right conjunct: NIL

Inference rules are applied by executing the attached commands. In general, it is necessary to provide some arguments for the application of a rule, which can be specified inside a generic command window. The command window adjusts itself automatically to the number of required arguments and provides some help for the requested parameters. The user can then specify the arguments either by manually entering them or by referring to certain nodes with a mouse-click.

In order to provide further support for interactive proof development, ΩMEGA uses a multi-layered focusing technique to compute suitable default values for rule applications [BS98]. These default values are suggested to the user as arguments in the command window.

Planner ΩMEGA's proof planner is based on an extension of the well-known STRIPS algorithm. It constructs a proof plan for a node g (the *goal node*) from a set I of *supporting nodes* (the initial state) using a set *Ops* of proof planning operators, called methods. The plans found by this procedure can be incorporated into the $\mathcal{PDS}$ as a separate level of abstraction. Furthermore, the proof planner also stores the reasons for its decisions for later use in proof explanation and analogy.

The ΩMEGA commands for evoking the planner and changing some settings relevant to the planner are provided by $\mathcal{L}\Omega\mathcal{UI}$ as menu items, such as applying the planner step by step, to do a certain number of planning steps, or to change the list of the proof operators (methods) considered by the planner. When the planner succeeds to find a plan, one can apply this plan to the $\mathcal{PDS}$. The graphical representation of the resulted $\mathcal{PDS}$ in $\mathcal{L}\Omega\mathcal{UI}$ shows the proof part of the $\mathcal{PDS}$ constructed by the planner.

In the near future, we intend to extend the planner so that it can be run in a reactive modus, i.e. reacting to user suggestions, such as to consider a given task next, or to take back some planner decision, and to continue with the next possible alternative. For this, the graphical representation of the $\mathcal{PDS}$ in $\mathcal{L}\Omega\mathcal{UI}$ must reflect the progress of the planner. Furthermore, the current agenda of planning goals must be displayed in parallel. This extension is facilitated by the client-server architecture (see section 4) of ΩMEGA that allows the user to enter suggestions to the planning process asynchronically with the help of appropriate Po-pup-menus.

External Systems - Automated Theorem Provers and Computer Algebra Systems ΩMEGA employs several automated theorem provers and computer algebra systems (for details cf. [KKS98]) as modules that can be applied to special-purpose proof problems. ΩMEGA uses for example OTTER [McC94], an automated theorem prover based on first order clause set resolution. We have described the integration of OTTER in [HKK+94] and the proof transformation necessary for incorporation of the result into the $\mathcal{PDS}$ in [HF96]; the methods described there also apply for the other theorem provers (SPASS, PROTEIN, and LEO) available in ΩMEGA. These systems can prove first-order theorems using various flags that control the search and the proof strategies. If for instance OTTER is called from ΩMEGA, some of these flags are set automatically, but others must be set by the user individually every time he uses OTTER.

To set these flags directly in ΩMEGA is laborious because it is necessary to know all valid values. $\mathcal{L}\Omega\mathcal{UI}$ provides an input mask that contains all flags with short descriptions of their valid values and what they will affect. Additionally, $\mathcal{L}\Omega\mathcal{UI}$ stores the last settings and offers it as a default value in the next call. $\mathcal{L}\Omega\mathcal{UI}$ controls several such "computational modules" by mapping their interface functionality into flexible input masks.

4 The Client-Server Architecture

A client-server architecture that separates ΩMEGA's logical kernel from its graphical user interface has increased its efficiency and maintainability.

In local computer networks the situation is quite common 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 proving theorems.

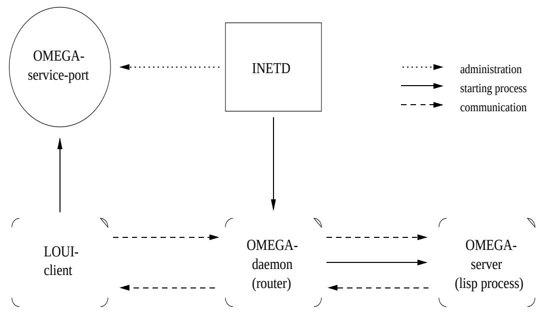
The maintenance advantage applies to both the user's and the developer's side. ΩMEGA is a rather large system (roughly 17 MB of COMMON LISP (CLOS) code for the main body in the current version), comprising numerous associated modules (such as the integrated automated theorem provers and a small computer algebra system) from different original sources, written in various programming languages. For the user 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 the current client-server architecture, the user only has to install the $\mathcal{L}\Omega\mathcal{UI}$ 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 $\mathcal{L}\Omega\mathcal{UI}$ is implemented in the OZ programming language [Saa98], which is freely available for various platforms, including UNIX and Windows95, this keeps the software and hardware requirements of the user moderate. The installation of the client is further simplified by the possibility of running $\mathcal{L}\Omega\mathcal{UI}$ as a Netscape applet, i.e. $\mathcal{L}\Omega\mathcal{UI}$ is automatically downloaded via the Internet. Thus we are able to provide current versions of ΩMEGA and $\mathcal{L}\Omega\mathcal{UI}$ without need for re-installation at the user's site.

Technical Realization $\mathcal{L}\Omega\mathcal{UI}$ is realized via a distributed programming system, called MOZART, which is an interactive distributed implementation of Oz. Mozart provides the full infrastructure to write distributed applications. Its main strength comes from its network transparency and network awareness.

Network transparency means that the semantics of Oz programs does not change if you distribute computations among different sites. For example, the programmer can use lexical scoping, logical variables, objects, etc. in distributed applications.

Network awareness means that the programmer has full control over the network operations. The language provides mobile and stationary objects, i.e. methods are executed locally (the object moves) or remotely (the message moves). The programmer has control over structure copying among sites. Structures may be copied eagerly or lazily. To reduce the bandwidth needed for communication, ΩMEGA implements an incremental approach based on SMALLTALK's MVC triad[1], which only transmits the parts of the $\mathcal{PDS}$ that are changed by a user action. This not only improves response times for low-bandwidth Internet connection but also focuses the user's attention to the effects of an action.



The Omega Client/Server Network A service called OMEGA is established on the server side, to which clients in form of $\mathcal{L}\Omega\mathcal{UI}$ applets can connect to. By addressing the port of the service an ΩMEGA daemon is started, the connection to the client is fulfilled and the main ΩMEGA LISP process comes up. This LISP process is also connected to the ΩMEGA daemon by an Internet socket. The administration and monitoring of the service's port is done by the Internet super server INETD, which listens for connections at certain Internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request. Therefore, each client using the OMEGA service, has its own ΩMEGA daemon and LISP process running. As mentioned the whole communication between the client and the server process is realized via Internet sockets using strings. The above figure illustrates the client-server architecture.

Since the presentation of the proof tree is defined by a context-free grammar, it should be easy to connect $\mathcal{L}\Omega\mathcal{UI}$ to different kind of provers. In this sense $\mathcal{L}\Omega\mathcal{UI}$ 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 the 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.

5 Related Work

User interfaces for theorem provers are credited with increasing importance in the field. These interfaces comprise graphical illustrations of proof structures and their elements, and facilities to set up commands in the proof environment.

Some special modes of proof types express part of the semantics of proof steps by graphical objects and annotations. Examples of this sort of visualization are binary decision diagrams for first-order deduction systems [PS95], 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 and by nested boxes enclosing segments of proof lines. Another presentation technique displays proof steps in an appropriately formatted and interactive way. [BJK+97] is able to present a proof in natural language, to a certain level of detail with deeper levels indented. In addition, levels of detail temporarily hidden can be exposed by clicking on the corresponding root proof line. A rather elaborate presentation system is CTCOQ [BKT94] which distributes the 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. Some other approaches put particular

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

emphasis on visualization by making the tree format of proof structures explicit in the display. The user interface for the SEAMLESS system [EM97] provides display facilities for a proof graph at different levels of abstraction in a framed window: a variety of lay-out operations including zooming and reuse of multiple appearances of lemmas. The user interface of INKA [HS96] allows for the display of induction proof sketches at varying levels of detail. Its features include status information, typically expressed by different coloring, and context-sensitive menus of possible user actions.

In comparison to these systems, ΩMEGA in some sense combines features of SEAMLESS and CTCOQ. 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 status information display is similar to that of CtCoq, but the database window is handled differently. Apart from that, the strict separation of visualizing the proof tree structure and browsing the terms associated with individual nodes selectively, handling of co-references, and the client-server architecture are unique features in OMEGA.

References

[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, *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

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

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

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

[BS98] Christoph Benzmüller and Volker Sorge. A Focusing Technique for Guiding Interactive Proofs. Submitted to the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications, 1998.

[EM97] J. Eusterbrock and N. Michalis. 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.

[HF96] Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in LNAI, pages 221–225, New Brunswick, NJ, USA, 1996. Springer Verlag.

[HF97] Xiaorong Huang and Armin Fiedler. Proof verbalization as an application of NLG. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, Nagoya, Japan, 1997. Morgan Kaufmann.

[HKK+94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Daniel Nesmith, Jörn Richts, and Jörg Siekmann. Ω-MKRP a proof development environment. In Alan Bundy, editor, *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI, pages 788–792, Nancy, France, 1994. Springer Verlag.

[HS96] D. Hutter and C. Sengler. A Graphical User Interface for an Inductive Theorem Prover. In *International Workshop on User Interface Design for Theorem Proving Systems*, 1996.

[KKS98] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 1998. Special Issue on the Integration of Computer Algebra and Automated Deduction; forthcoming.

[McC94] W. W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.

[PS95] J. Posegga and K. Schneider. Interactive First-Order Deduction with BDDs. In *International Workshop on User Interface Design for Theorem Proving Systems*, Glasgow, 1995.

[Saa98] Programming Systems Lab Saarbrücken, 1998. The Oz Webpage: `http://www.ps.uni-sb.de/ns3/oz/`.