PLATΩ: A Mediator between Text-Editors and Proof Assistance Systems

Marc Wagner

Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany (`www.ags.uni-sb.de/~mwagner`)

Serge Autexier

DFKI GmbH & Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany (`www.dfki.de/~serge`)

Christoph Benzmüller

Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany (`www.ags.uni-sb.de/~chris`)

Abstract

We present a generic mediator, called PLATΩ, between text-editors and proof assistants. PLATΩ aims at integrated support for the development, publication, formalization, and verification of mathematical documents in a natural way as possible: The user authors his mathematical documents with a scientific WYSIWYG text-editor in the informal language he is used to, that is a mixture of natural language and formulas. These documents are then semantically annotated preserving the textual structure by using the flexible, parameterized proof language which we present. From this informal semantic representation PLATΩ automatically generates the corresponding formal representation for a proof assistant, in our case ΩMEGA. The primary task of PLATΩ is the maintenance of consistent formal and informal representations during the interactive development of the document.

Keywords: text-editor, proof-assistance systems, user interface, ΩMEGA, Texmacs

1 Introduction

Unlike computer algebra systems, mathematical proof assistance systems have not yet achieved considerable recognition and relevance in mathematical practice. Clearly, the functionalities and strengths of these systems are generally not sufficiently developed to attract mathematicians on the edge of research. For applications in e-learning and engineering contexts their capabilities are often sufficient, though. However, even for these applications significant progress is still required,

in particular with respect to the usability of these systems. One significant short-coming of the current systems is that they are not fully integrated into or accessible from within standard mathematical text-editors.

For purposes such as tutoring mathematics, communicating or publishing mathe-matical documents, the content is in practice usually encoded using common mathe-matical representation languages by employing standard mathematical text editors. Proof assistance systems, in contrast, require fully formal representations and are not yet sufficiently linked with these standard mathematical text-editors. There-fore, rather than developing a new user interface for the mathematical assistance system ΩMEGA [24], we propose a generic way of extending ΩMEGA to serve as a mathematical service provider for scientific text-editors.

'If the mountain won't come to Mohammed, Mohammed must go to the mountain.'

Our approach allows the user to write his mathematical documents in the lan-guage he is used to, that is a mixture of natural language and formulas. These documents can then be semantically annotated preserving the textual structure by using the flexible parameterized proof language we present. From this semantic rep-resentation PLATΩ automatically builds up the corresponding formal representation in ΩMEGA and takes further care of the maintenance of consistent versions.

The formal representation allows the underlying proof assistance system to sup-port the user in various ways, including the management of mathematical defini-tions, theorems and proofs, as well as the access to automatic theorem provers, computer algebra systems, and other mathematical tools in order to automatically verify conclusions and computations made by the user and to suggest possible cor-rections. These functionalities can be provided through PLATΩ by context-sensitive service menus in order to support the interactive development of mathematical doc-uments at a high level of abstraction.

On the one hand, these services could include the possibility to automatically generate parts of the proof as well as computations in order to disburden the user of taking care about cumbersome details and to let him concentrate on the substantial parts of the proof. Thus, menu interaction may lead to changes of the formal rep-resentation which are reflected by PLATΩ in changes of the semantic representation in the document. On the other hand, further proof development in the text-editor leads to changes in the document which are propagated by PLATΩ to changes in the formal representation in ΩMEGA.

Altogether, this approach allows for the incremental, interactive development of mathematical documents which in addition can be formally validated by ΩMEGA, hence obtaining *verified mathematical documents*. This approach is generally inde-pendent of the proof assistance system as well as the text-editor. Nevertheless the scientific WYSIWYG text-editor TEX_MACS [27] provides professional type-setting and supports authoring with powerful macro definition facilities like in LATEX. It moreover allows for the definition of plug-ins that automatically process the docu-ment and is thus especially well-suited for an integration of PLATΩ.

This paper is organized as follows: Section 2 presents an overview on the ΩMEGA

system in order to more concretely motivate our setting. Section 3 introduces the mediator PLATΩ with a focus on the interfaces to the text-editor and the proof assistance system. A working example is presented in Section 4 that illustrates the integration of PLATΩ into a scientific text editor like for example TEX_{MACS}. The paper concludes with an overview on related work (Section 5) and a summary of the major results in Section 6.

2   Preliminaries: Ωmega, Maya, and the Task Layer

The development of the proof assistance system ΩMEGA is one of the major attempts to build an all encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof

construction for domains with rich and well-structured mathematical knowledge (see Figure on the right). The ΩMEGA-system is currently under re-development where, among others, it is augmented by the development graph manager MAYA and the underlying natural deduction calculus is replaced with the CORE-calculus [4].

The MAYA system [8] supports an evolutionary formal development by allowing users to specify and verify developments in a structured manner, it incorporates a uniform mechanism for verification in-the-large to exploit the structure of the specification, and it maintains the verification work already done when changing the specification. Proof assistance systems like ΩMEGA rely on mathematical knowledge formalized in structured theories of definitions, axioms and theorems. The MAYA system is the central component in the new ΩMEGA system that takes care about the management of change of these theories via its OMDOC-interface [19].

The CORE-calculus supports proof development directly at the *assertion level* [17], where proof steps are justified in terms of applications of definitions, axioms, theorems or hypotheses (collectively called *assertions*). It provides the logical basis for the so-called TASK LAYER [14], that is an instance of the new proof datastructure (PDS) [5]. The TASK LAYER is the central component for computer-based proof construction in ΩMEGA. It offers a uniform proof construction interface to both the human user and the automated proof search procedures MULTI [21] and ΩANTS [9,26]. The nodes of the PDS are annotated with *tasks*, which are Gentzen-style multi-conclusion sequents augmented by means to define multiple foci of attention on subformulas that are maintained during the proof. Each task is reduced to a possibly empty set of subtasks by one of the following proof construction steps: (1) the introduction of a proof sketch [30] [1] , (2) deep structural rules for weakening and decomposition of subformulas, (3) the application of a lemma that can be pos-

---
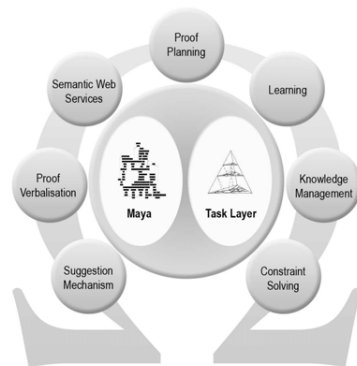
[1] In the old ΩMEGA system this was realized by using so-called *Island*-methods.

tulated on the fly, (4) the substitution of meta-variables, and (5) the application of an inference. Inferences are the basic reasoning steps of the TASK LAYER, and comprise assertion applications, proof planning methods or calls to external theorem provers or computer algebra systems (see [14,6] for more details about the TASK LAYER).

A formal proof requires to break down abstract proof steps to the CORE calculus level by replacing each abstract step by a sequence of calculus steps. This has usually the effect that a formal proof consists of many more steps than a corresponding informal proof of the same conjecture. Consequently, if we manually construct a formal proof many interaction steps are typically necessary. Formal proof sketches [30] in contrast allow the user to perform high level reasoning steps without having to justify them immediately. The underlying idea is that the user writes down only the interesting parts of the proof and that the gaps between these steps are filled in later, ideally fully automatically (see also [24]). Proof sketches are thus a highly relevant for a mediator like PLATΩ whose task it is to support the transition to fully formal representations from an informal proof in a mathematical document via intermediate representations of underspecified proof sketches.

3   The PLATΩ System

The mediator PLATΩ is designed to run either locally as a plugin for a particular text-editor or as a mathematical service provider which text editors could access through the web. In order to manage different text-editor clients as well as different documents in the same client, we integrated session management into PLATΩ. The text-editor may request a unique session key which it has to provide as an argument for any further interaction in this particular session.

PLATΩ is connected with the text-editor by an informal representation language which flexibly supports the usual textual structure of mathematical documents. Furthermore, this semantic annotation language, called _proof language_ (PL), allows for underspecification as well as alternative (sub)proof attempts. In order to generate the formal counterpart of a PL representation, PLATΩ separates theory knowledge like definitions, axioms and theorems from proofs. The theories are formalized in the _development graph language_ (DL), which is close to the OMDOC theory language supported by the MAYA system, whereas the proofs are transformed into the _tasklayer language_ (TL) which describes the PDS instance of the TASK LAYER. Hence, PLATΩ is connected with the proof assistance system ΩMEGA by a formal representation close to its internal datastructure.

Besides the transformation of complete documents, it is essential to be able to propagate small changes from an informal PL representation to the formal DL/TL one and the way back. If we always perform a global transformation, we would on the one hand rewrite the whole document in the text-editor which means to lose large parts of the natural language text written by the user. On the other hand we would reset the datastructure of the proof assistance system to the abstract level of proof sketches. For example, any already developed expansion towards calculus

Figure 1. PLATΩ mediates between natural mathematical texts and the proof assistant ΩMEGA

level or any computation result from external systems would be lost. Therefore, one of the most important aspects of PLATΩ's architecture is the propagation of changes.

The formal representation finally allows the underlying proof assistance system to support the user in various ways. PLATΩ provides the possibility to interact through context-sensitive service menus. If the user selects an object in the document, PLATΩ requests service actions from the proof assistance system regarding the formal counterparts of the selected object. Hence, the mediator needs to maintain the mapping between objects in the informal language PL and the formal languages DL and TL.

In particular, the proof assistance system could support the user by suggesting possible inference applications for a particular proof situation. Since the computation of all inference argument instantiations may take a long time, a multi-level menu with the possibility of lazy evaluation is required. PLATΩ supports the execution of nested actions inside a service menu which may result in a change description for this menu.

Through service menus the user may get access to automatic theorem provers and computer algebra systems which could automatically verify conclusions and computations and suggest possible corrections. These and many more functionalities are supported by PLATΩ through its mechanism to propagate changes as well as the possibility of custom answers to the user of the text-editor. Altogether, the mediator PLATΩ is designed to support the interactive development of mathematical documents at a high level of abstraction.

3.1 PlatΩ's Interfaces

PLATΩ provides abstract interfaces to the text-editor and the proof assistance system (see also Fig. 1). Before we discuss their design and realization, we first present the functionalities of PLATΩ from the perspective of the text-editor. PLATΩ's methods are:

- **Initialize a session**: `plato:init` starts a new session in PLATΩ

- **Upload a document**: `plato:upload` uploads a whole document in the informal language PL, from which PLATΩ builds up the formal representations DL and TL. If a document has already been uploaded, PLATΩ performs an internal difference analysis using a semantic based differencing mechanism [22] and then proceeds as with patching the document.

- **Patch a document**: `plato:patch` patches an already uploaded document in the informal language PL with patch information given in the XUPDATE standard (see Section 3.2). PLATΩ transforms this patch information into patches for the formal representations DL and TL, which are used to patch the datastructure of the proof assistance system.

- **Request a menu**: `plato:service` requests a menu for an object in the informal language PL inside the document. The response is either a menu in the service language SL (or an error message). PLATΩ uses its maptable relating objects in PL with objects in DL and TL to requests service support from the proof assistance system for the latter.

- **Execute a menu action**: `plato:execute` triggers the execution of an action with its actual arguments. The result can be a patch for the current menu, a patch for the document or a custom answer (or an error message). The purpose is to evaluate an action inside a menu. This style of responses offers quite many interaction possibilities: If the selected action was nested somewhere in the menu, the proof assistance system will usually modify the menu. This will be propagated by PLATΩ to a corresponding response which only modifies the menu and leaves the patch for the document and the custom answer empty. If the selected action was situated on top level of the menu, the execution in the proof assistance system will more likely change the formal representation. Anyhow, PLATΩ propagates these changes to changes in the informal presentation of the text-editor, such that the response will usually remove the menu and patch the document appropriately. The custom answer leaves room for arbitrary interaction possibilities like knowledge retrieval or natural language feedback.

- **Close a session**: `plato:close` terminates a session.

A detailed descriptions of PLATΩ's interface functions is given in Appendix A.

3.2 Interface to the Text-Editor and Proof Assistance System

The goal of PLATΩ is to lay a compatible foundation for a text-editor interface across different environments. It should be a clean, extensible interface that is very simple and easy to implement such that it could quickly be adapted to run with any scientific text editor on any operating system. Therefore we decided to represent the mathematical document as well as the service menus in XML [11], the patches for documents and menus in the XUPDATE update language [20] and to use XML-RPC [31] as interface protocol. XUPDATE [20] is an XML update language which uses XML to encode its updates and the expression language XPATH [10] to select elements for processing. An update may contain the following types of elements relevant for PLATΩ: `insert-before`, `insert-after`, `append`, `update`, `remove`. All

operations in an update have to be applied in parallel to the target document. XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. It is a very simple protocol, defining only a handful of data types and commands, and its entire two page description can be found at [31].

The ΩMEGA system is implemented in LISP. Therefore, we decided to implement the interface to ΩMEGA, which provides LISP functions for each PLATΩ method, in LISP too. These functions operate only on the formal representation of the mathematical document and they will be illustrated in more detail in the next Section. PLATΩ allows to start, stop and manage multiple servers in parallel for the same proof assistance instance. Generally, we aim at an approach that is independent of the particular proof assistance system to be integrated. Therefore the proof language as well as the service menu language are parameterized over the sublanguages for definitions, formulas, references and menu argument content. Extending these sublanguages allows to scale up the power of the whole system regarding representation capabilities as well as service functionalities. As soon as there will be significant progress in the area of natural language analysis, one could even allow full natural language in these sublanguages. Thus PLATΩ is designed to support the evolution of the underlying proof assistance system towards an ideal mathematical assistance system. We will present some more aspects of this more general viewpoint in the next Section. The focus, however, is on the integration of the ΩMEGA system into the scientific WYSIWYG text-editor TEX_MACS.

4 A Working Example

In this section we will evaluate the mediator PLATΩ in combination with ΩMEGA and TEX_MACS. We will illustrate all available methods of PLATΩ by discussing a working example in the theory of Simple Sets.

In this paper, we describe the mediation between the informal representation in the text-editor and the formal representation in the proof assistance system on an abstract level. All details on the communicated documents, patch descriptions and menus for this example can be found in [28].

Since the TEX_MACS interface for proof assistance systems is under continuous development, a PLATΩ plugin for TEX_MACS has been developed by the ΩMEGA group that maps the interface functions of PLATΩ to the current ones of TEX_MACS and which defines a style file for PL macros in TEX_MACS. In the following example, we use this plugin to establish a connection between TEX_MACS and PLATΩ's XML-RPC server.

First of all, the text-editor TEX_MACS initializes a new session by calling the method `plato:init` together with a client name, for example "`texmacs#1`". The resulting session name has to be saved by the text-editor in order to use it for the following communication with PLATΩ.
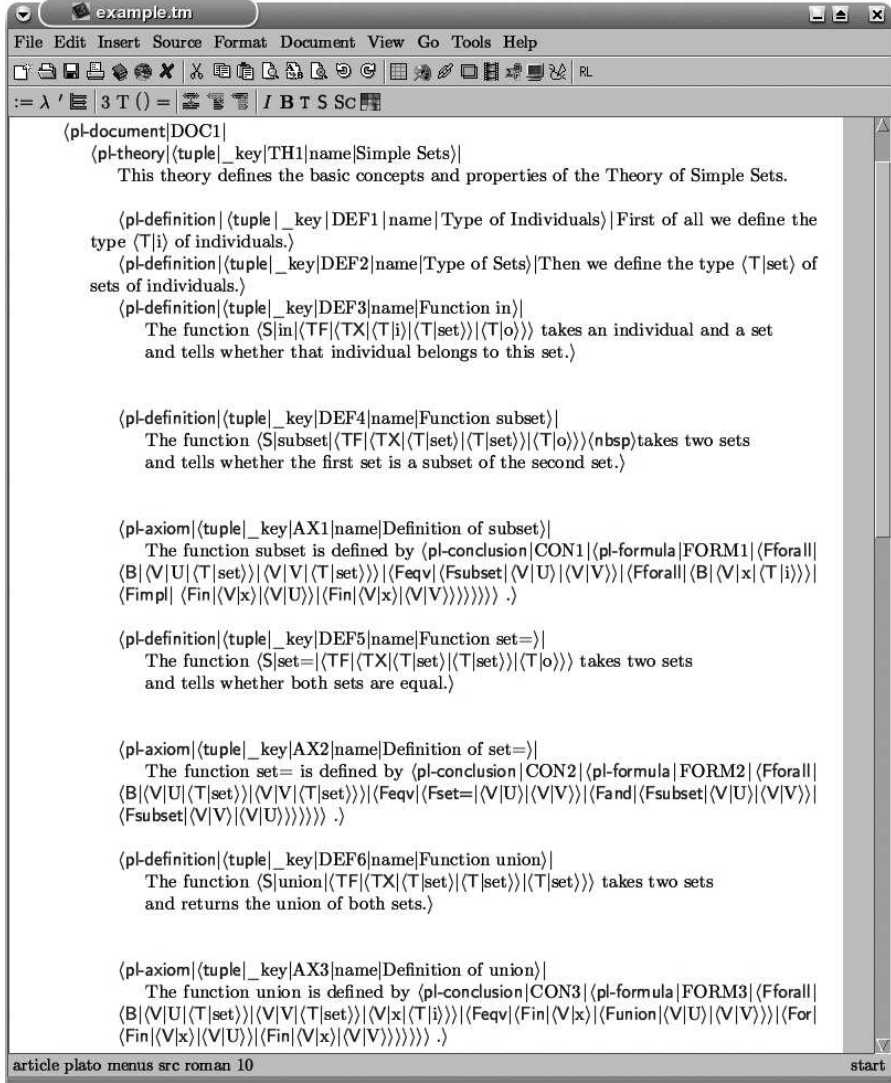
In the text-editor, we have written an example document with the semantic annotation language PL (defined in [28]). The theory *Simple Sets* in this document
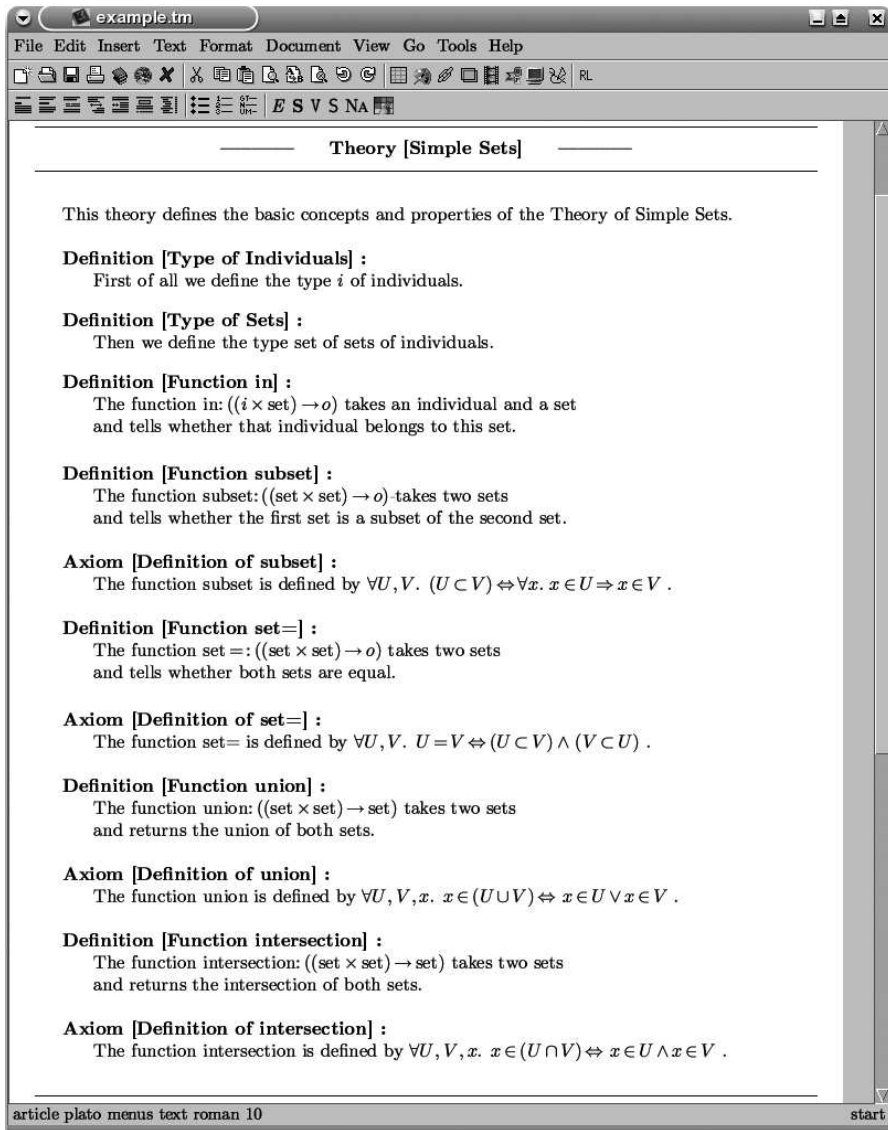
Figure 2. Theory *Simple Sets* in TeX_MACS

contains for example definitions and axioms for *subset*, *set=*, *union* and *intersection*. Fig. 2 shows the theory as seen in TeX_MACS and Fig. 3 shows the encoding of this theory in TeX_MACS with PLATΩ macros.

Furthermore, we have written a theory *Distributivity in Simple Sets* which imports all knowledge from the first theory *Simple Sets*. This second theory consists of a theorem about the *Distributivity of intersection*. The user has already started a proof for this theorem by introducing two subgoals. Fig. 4 shows the theory as seen in TeX_MACS and Fig. 5 shows the encoding of this theory in TeX_MACS.

By pressing a keyboard shortcut, the user can always easily switch between both views in the text-editor. The PL macros contained in the document must be

example.tm

File　Edit　Insert　Source　Format　Document　View　Go　Tools　Help

⟨pl-document|DOC1|
　⟨pl-theory|⟨tuple|_key|TH1|name|Simple Sets⟩|
　　This theory defines the basic concepts and properties of the Theory of Simple Sets.

　　⟨pl-definition|⟨tuple|_key|DEF1|name|Type of Individuals⟩|First of all we define the
type ⟨T|i⟩ of individuals.⟩
　　⟨pl-definition|⟨tuple|_key|DEF2|name|Type of Sets⟩|Then we define the type ⟨T|set⟩ of
sets of individuals.⟩
　　⟨pl-definition|⟨tuple|_key|DEF3|name|Function in⟩|
　　　The function ⟨S|in|⟨TF|⟨TX|⟨T|i⟩|⟨T|set⟩⟩|⟨T|o⟩⟩⟩ takes an individual and a set
　　　and tells whether that individual belongs to this set.⟩

　　⟨pl-definition|⟨tuple|_key|DEF4|name|Function subset⟩|
　　　The function ⟨S|subset|⟨TF|⟨TX|⟨T|set⟩|⟨T|set⟩⟩|⟨T|o⟩⟩⟩⟨nbsp⟩takes two sets
　　　and tells whether the first set is a subset of the second set.⟩

　　⟨pl-axiom|⟨tuple|_key|AX1|name|Definition of subset⟩|
　　　The function subset is defined by ⟨pl-conclusion|CON1|⟨pl-formula|FORM1|⟨Fforall|
⟨B|⟨V|U|⟨T|set⟩⟩|⟨V|V|⟨T|set⟩⟩⟩|⟨Feqv|⟨Fsubset|⟨V|U⟩|⟨V|V⟩⟩|⟨Fforall|⟨B|⟨V|x|⟨T|i⟩⟩|
⟨Fimpl|⟨Fin|⟨V|x⟩|⟨V|U⟩⟩|⟨Fin|⟨V|x⟩|⟨V|V⟩⟩⟩⟩⟩⟩⟩⟩ .⟩

　　⟨pl-definition|⟨tuple|_key|DEF5|name|Function set=⟩|
　　　The function ⟨S|set=|⟨TF|⟨TX|⟨T|set⟩|⟨T|set⟩⟩|⟨T|o⟩⟩⟩ takes two sets
　　　and tells whether both sets are equal.⟩

　　⟨pl-axiom|⟨tuple|_key|AX2|name|Definition of set=⟩|
　　　The function set= is defined by ⟨pl-conclusion|CON2|⟨pl-formula|FORM2|⟨Fforall|
⟨B|⟨V|U|⟨T|set⟩⟩|⟨V|V|⟨T|set⟩⟩⟩|⟨Feqv|⟨Fset=|⟨V|U⟩|⟨V|V⟩⟩|⟨Fand|⟨Fsubset|⟨V|U⟩|⟨V|V⟩⟩|
⟨Fsubset|⟨V|V⟩|⟨V|U⟩⟩⟩⟩⟩⟩⟩ .⟩

　　⟨pl-definition|⟨tuple|_key|DEF6|name|Function union⟩|
　　　The function ⟨S|union|⟨TF|⟨TX|⟨T|set⟩|⟨T|set⟩⟩|⟨T|set⟩⟩⟩ takes two sets
　　　and returns the union of both sets.⟩

　　⟨pl-axiom|⟨tuple|_key|AX3|name|Definition of union⟩|
　　　The function union is defined by ⟨pl-conclusion|CON3|⟨pl-formula|FORM3|⟨Fforall|
⟨B|⟨V|U|⟨T|set⟩⟩|⟨V|V|⟨T|set⟩⟩|⟨V|x|⟨T|i⟩⟩⟩|⟨Feqv|⟨Fin|⟨V|x⟩|⟨Funion|⟨V|U⟩|⟨V|V⟩⟩⟩|⟨For|
⟨Fin|⟨V|x⟩|⟨V|U⟩⟩|⟨Fin|⟨V|x⟩|⟨V|V⟩⟩⟩⟩⟩⟩⟩ .⟩

article plato menus src roman 10　　　start

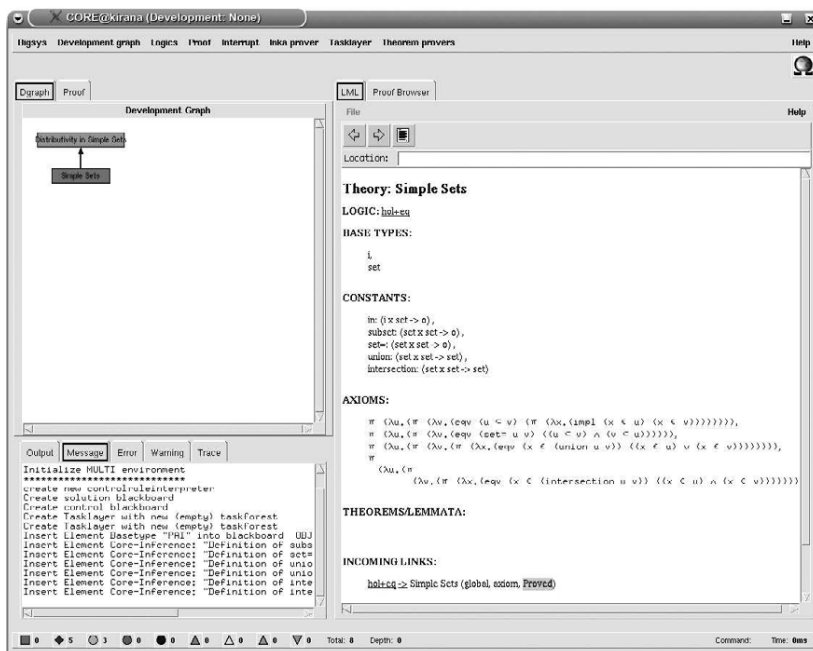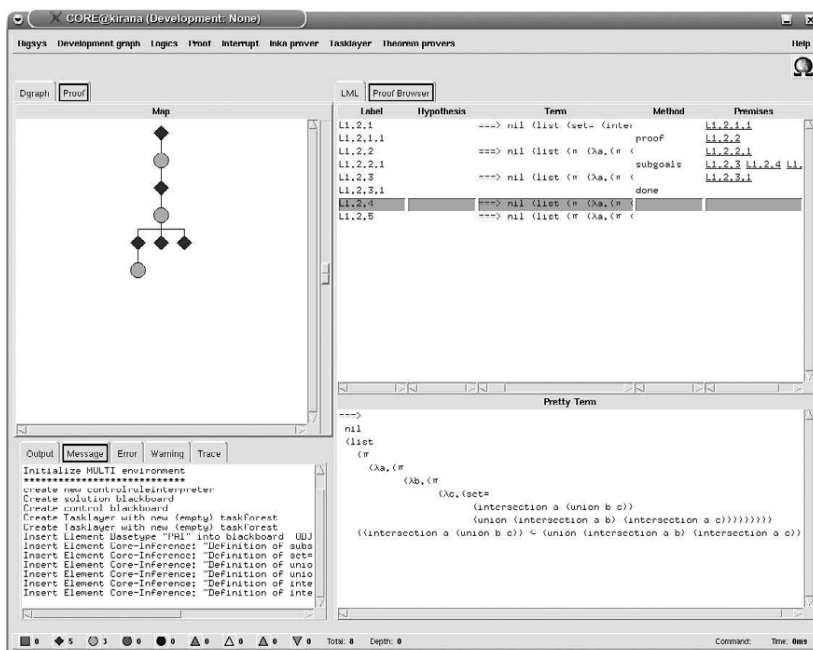Figure 3. Encoding of Theory *Simple Sets* in T$_E$X$_{MACS}$

provided by the user [2] and are used to automatically extract the corresponding PL document, the informal representation of the document for PLATΩ.

Uploading this PL document with `plato:upload`, PLATΩ separates theory knowledge like definitions, axioms and theorems from proofs and starts generating the formal representation.

On the one hand, PLATΩ creates a DL document containing definitions, axioms and theorems in a representation close to OMDOC. On the other hand, the proof is transformed into a TL document, an abstract representation for the PDS instance of the TASK LAYER in the proof assistance system.

From the DL document, the PLATΩ instance for ΩMEGA generates a theory

---

[2] Currently this still requires some expertise about PL and the T$_E$X$_{MACS}$ macro language. Future work includes to provide better support for this task.

Figure 4. Theory *Distributivity in Simple Sets* in $\TeX_{MACS}$

Figure 5. Encoding of Theory *Distributivity in Simple Sets* in $\TeX_{MACS}$

representation in OMDOC that MAYA takes as input for the creation of a development graph. Fig. 6 shows the theories uploaded in ΩMEGA. For this evaluation we use the old user interface LΩUI [25] to visualize the status of ΩMEGA. The user interacts of course only with the text-editor. The old LΩUI interface, including the display of MAYA's development graphs, shall be entirely replaced by $\TeX_{MACS}$ and PLATΩ. They are only presented in this paper to show the internal representation obtained from $\TeX_{MACS}$ via PLATΩ. From the TL document, the PLATΩ instance for ΩMEGA builds up the concrete datastructure of the TASK LAYER (see Fig. 7).

The upload procedure has terminated successfully with the complete generation of the formal representation in the proof assistance system, hence PLATΩ returns "OK".

Figure 6. Theory *Simple Sets* in ΩMEGA

Figure 7. Partial Proof of Theorem *Distributivity of intersection* in ΩMEGA

Further developing the document, the user has started to prove the first subgoal by deriving a new subgoal and introducing an assumption (see Fig. 8). This modification of the encoding of the document (see Fig. 9) has to be propagated by PLATΩ to the formal representation in ΩMEGA. In general, the difference with respect to
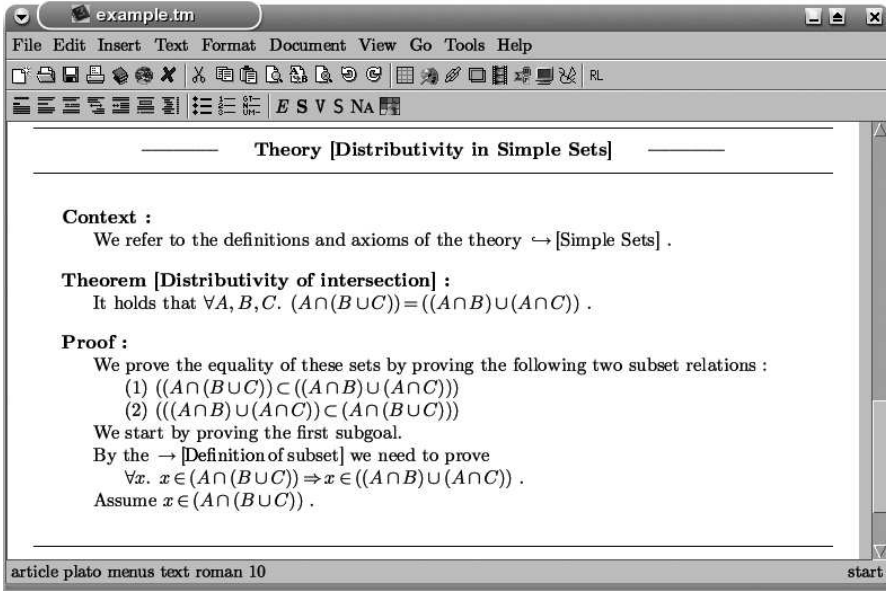
Figure 8. Modification of the Proof in $\TeX_{MACS}$ by the User
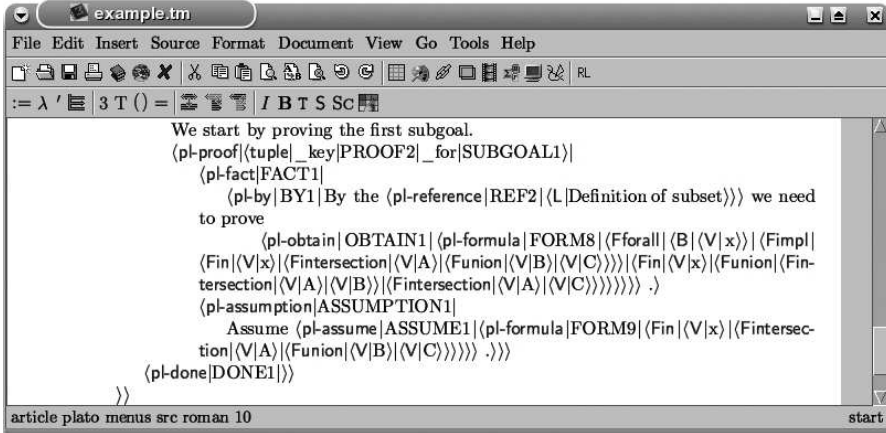
Figure 9. Modification of the Encoding of the Proof in $\TeX_{MACS}$ by the User

the last synchronized version of the document should be computed and send to PLATΩ by using `plato:patch`. At the moment, $\TeX_{MACS}$ is not able to compute this difference, therefore the whole document is send again by `plato:upload` and PLATΩ computes the difference.

The difference of the informal PL document is then transformed by PLATΩ to a difference of the formal representations in DL and TL. Since the modifications do not affect theory knowledge, this transformation only results in modifications for the intermediate representation and finally the representation of the TASK LAYER proof data structure.

The PLATΩ instance for ΩMEGA uses this patch information to modify the TASK LAYER rather than to completely rebuild it from scratch (see Fig. 10). The patch
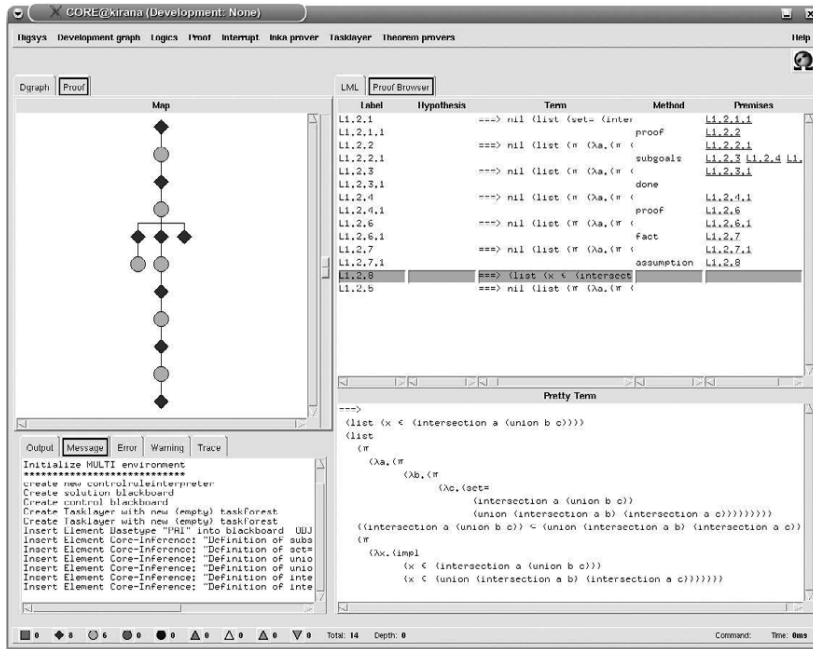
Figure 10. Modification of the Proof in ΩMEGA by PLATΩ
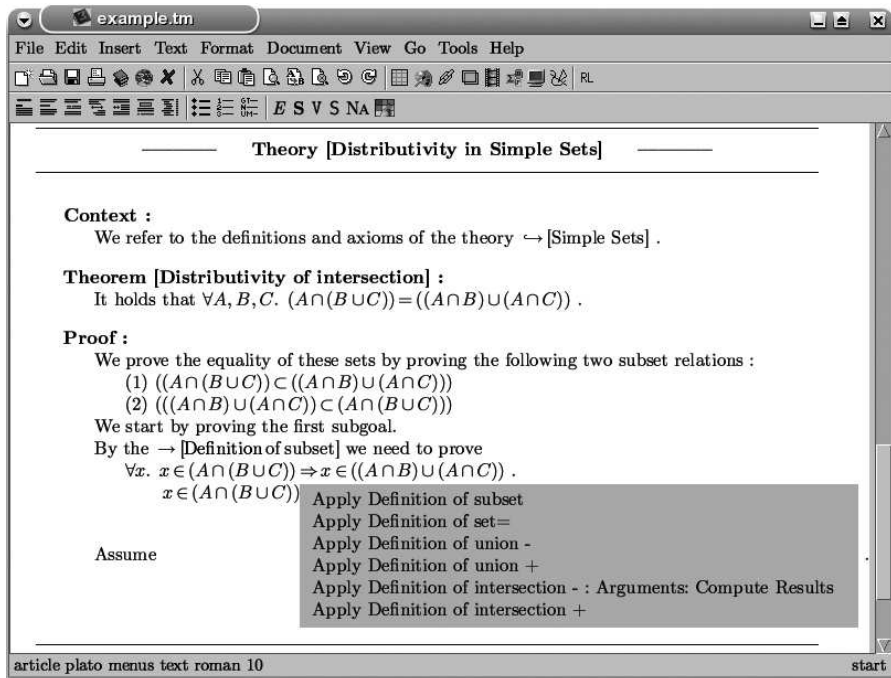
Figure 11. Service Menu in TEX_MACS requested by the User

procedure has terminated successfully, hence PLATΩ returns "OK". Altogether, the user is able to synchronize his informal representation in the text-editor document with the formal representation in the proof assistance system.
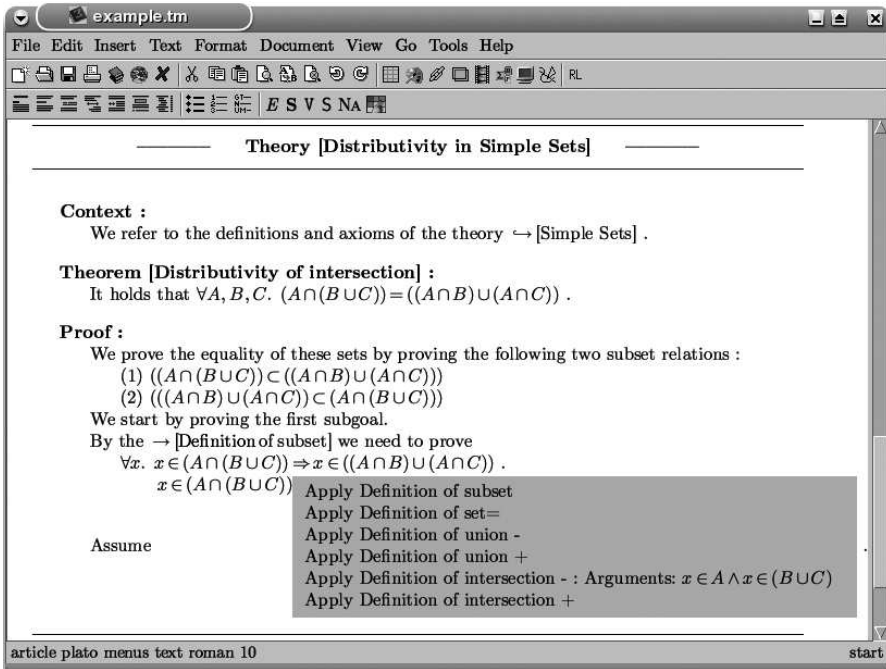
Figure 12. Modification of the Service Menu in T_EX_{MACS} by PLATΩ
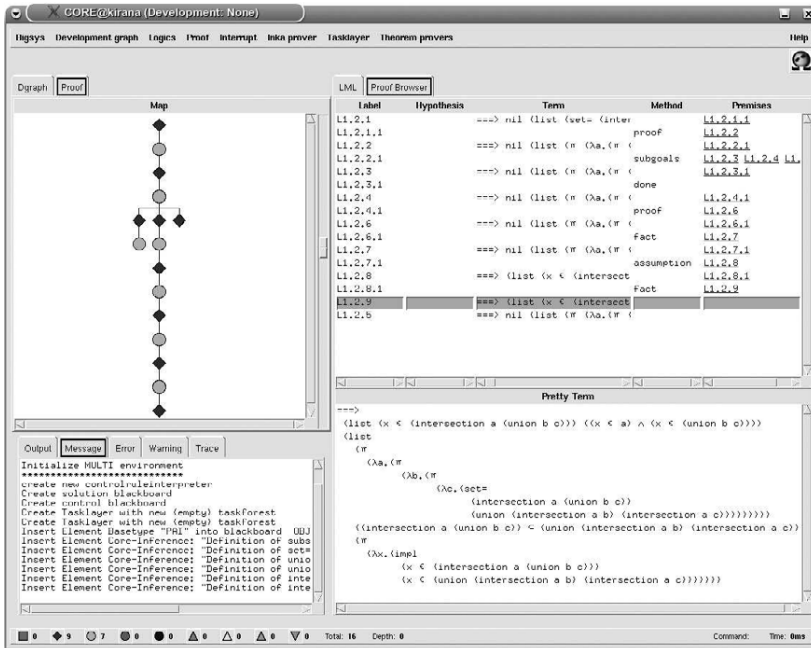
Figure 13. Modification of the Proof in ΩMEGA by the System

The next interesting feature of PLATΩ is the possibility of getting system support from the underlying proof assistance system. Selecting the recently introduced formula in the assumption, the user requests a service menu from PLATΩ.
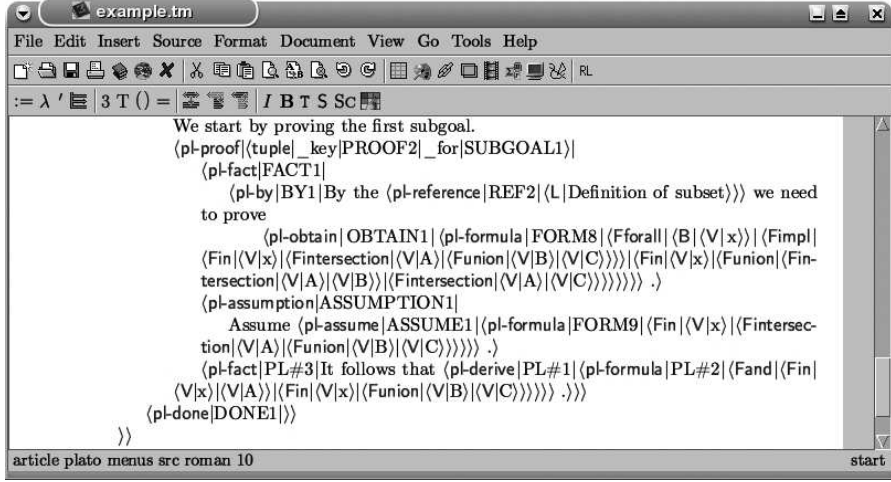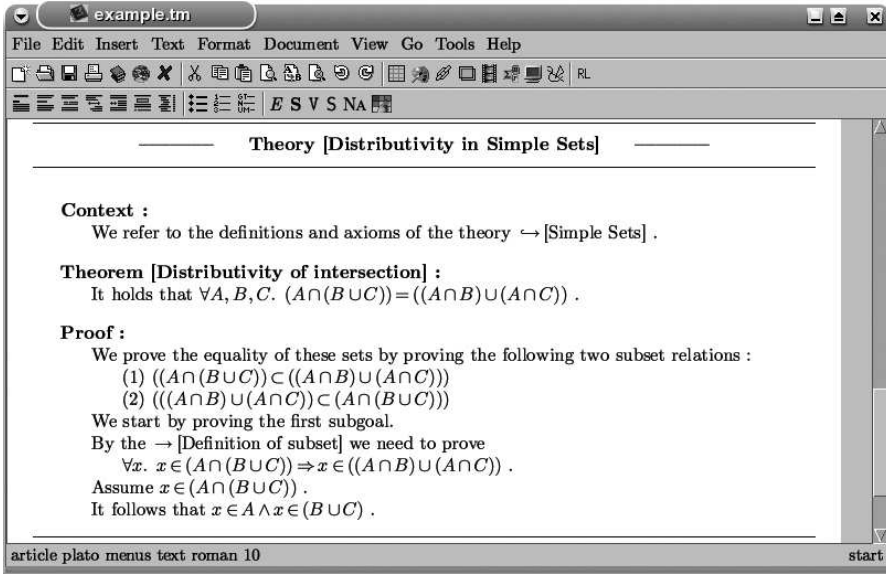
Figure 14. Modification of the Encoding of the Proof in T$_{\text{E}}$X$_{\text{MACS}}$ by PLATΩ

Requesting services for the corresponding task in the TASK LAYER, a list of available inferences is returned to PLATΩ. In order to answer quickly to the text-editor, we generate nested actions that allow to incrementally compute the formulas resulting from the application of an inference rather than to precompute all possible resulting formulas for all available inferences. For this example, the inferences were manually generated in the proof assistance system, since the automatic inference generation from the theory knowledge in the development graph is still under development.

The menu is displayed to the user in T$_{\text{E}}$X$_{\text{MACS}}$ as shown in Fig. 11, where we already expanded the action **Apply Definition of intersection -** to its nested action **Compute Results**. Executing **Compute Results** calls the method `plato:execute` in PLATΩ, which leads in the TASK LAYER to the computation of all resulting formulas for the inference **Definition of intersection -**, defined by the corresponding axiom. PLATΩ tells the text-editor how to change the menu by sending a patch description for the menu.

The user selects the desired formula (see Fig. 12) which triggers the application of the top level inference and launches a `plato:execute`. PLATΩ calls the TASK LAYER for the application of the selected inference in order to obtain the chosen formula. The TASK LAYER performs the requested operation which typically modifies the proof data structure (see Fig. 13). This modification is transformed by the PLATΩ instance for ΩMEGA into a patch description for the formal representation in TL.

After that, PLATΩ transforms this TL patch into an IL patch and finally a PL patch for the informal document in T$_{\text{E}}$X$_{\text{MACS}}$, which is then send to the text-editor. Furthermore, the menu is closed by sending a patch description which removes it. Currently, the new proof fragments are inserted together with additional predefined natural language fragments. However, we plan to integrate the natural language proof presentation system P.REX [15] into PLATΩ, in order to generate true natural language output for the proof steps added by the proof assistance system.

Figure 15. Modification of the Proof in T$_{\!E}$X$_{\text{MACS}}$ by PLATΩ

The text-editor finally patches the encoding of the document (see Fig. 14) according to this patch description. Fig. 15 shows the patched document displayed in T$_{\!E}$X$_{\text{MACS}}$.

Note that the user can change any part of the document, including the parts generated by the proof assistance system. Due to the maintenance of consistent versions, the further development of the document can be a mix of manual authoring by the user and interactive authoring with the proof assistance system.

Last but not least, closing the document or the text-editor will close the active session in PLATΩ and in the proof assistance system ΩMEGA by calling the method `plato:close`. For this evaluation we chose a simple mathematical domain in order to focus on the system behavior of the mediator. In general, the problem solving capabilities only depend on the underlying proof assistance system. Cutting edge proof assistance can be provided by extending the representational sublanguages for complicated domains.

5　Related Work

The AUTOMATH project [13] of Nicolas de Bruijn and his idea of a mathematical vernacular has to be mentioned as pioneering work in the field. Similar to AUTOMATH, the MIZAR [3] and ISAR [29] projects aim at a well balanced compromise between rigorous, formal representation languages suitable for machine processing and human readable, natural representations. The "grammatical framework" approach (GF) [23] goes one step further and employs a λ-calculus based formalism to define grammars consisting of an abstract and a concrete syntax. In the abstract

---

[3]　www.mizar.org

syntax one can formally represent mathematical definitions, theorems and proofs and check their correctness and the concrete syntax defines a mapping from the abstract syntax into linguistic objects. A common problem of these approaches are the diverging requirements of representation to the machine and the user side. AUTOMATH as well as MIZAR and ISAR sacrifice readability to obtain machine processability. GF in contrast shows high readability as well as machine processability but the supported fragment of natural language is far too small and inflexible to allow mathematicians to use their familiar language.

Many mathematical assistance systems favor machine processability over human authoring, while trying to enhance readability. This is done by separating input and output language: the input language remains machine oriented whereas the output language gets close to natural language. The system PCOQ [2] for example uses a schematic approach to represent its output in quasi-natural language. The systems NUPRL [16], CLAM [1] and ΩMEGA/P.REX [15] go further and use natural language processing techniques to generate true natural language output. THEOREMA [12] is a system which strictly separates informal and formal parts in mathematical documents: The user can input informal parts of text without any restriction but these parts are not used for machine processing. The formal parts, however, have to be written in the input language of the computer algebra system MATHEMATICA.

In contrast to that, we suggest in our approach [7] a formal representation language for mathematical content detached from any particular logic or calculus. This allows us to represent arbitrary content regardless of the underlying logic. Moreover, the language allows us to represent both different levels of concept and underspecification and is thus particularly well-suited to represent proofs that are authored in a natural way by human beings. Closely related to our approach is the MATH-LANG project [18]. It also proposes a top-down approach starting from natural mathematical texts towards machine processing. However, the MATHLANG project so far concentrates mainly on supporting the analysis of the abstract representations based on type checking and, in contrast to our approach, the gap between real theorem provers and mathematical assistance tools remains open.

To our knowledge there has not been any attempt to integrate a proof assistance system with text-editors in the flexible way as done via PLATΩ. All approaches described above do not consider the input document as an independent, first-class citizen with an internal state that has to be kept consistent with the formal representations in the proof assistance system while allowing arbitrary changes on each side. The only work in that direction has been carried in the context of PROOF GENERAL [3]. In PROOF GENERAL the user edits a central document in a suitable editing environment, from which it can be evaluated by various tools, such as a proof assistant, which checks whether the document contains valid proofs, or a renderer which typesets or renders the document into human oriented documentation readable outwith the system. However, the system is only an interface to proof assistance systems that process their input incrementally. Hence, the documents edited in PROOF GENERAL are processed incrementally in a top-down manner and especially parts that have been processed by the proof assistance systems are locked

and cannot be edited by the user. Furthermore, the documents are in the input format of the proof assistant rather than in the format of some type-setting program. Though we have tried to design the functionalities and representation languages in PLATΩ's interface as general as possible, future work will have to show that PLATΩ can be as easily adapted to different proof assistants as is already possible for PROOF GENERAL.

6   Conclusion

The main contribution is the design and development of a generic mediator, called PLATΩ, between text-editors and the proof assistance system ΩMEGA. The presented mediator allows the user to write his mathematical documents in the language he is used to, that is a mixture of natural language and formulas. These documents are semantically annotated preserving the textual structure by using a flexible parameterized proof language. PLATΩ automatically builds up the corresponding formal representation in ΩMEGA and takes further care of the maintenance of consistent versions while providing a mechanism to propagate changes between both representations. All kinds of services of the underlying proof assistance system regarding the formal representation can be provided through PLATΩ by context-sensitive service menus in order to support the interactive development of mathematical documents at a high level of abstraction. Altogether, PLATΩ contributes to the evolution of proof assistance systems towards ideal mathematical assistance systems.

In this paper we have illustrated how informal, natural proofs developed in the text-editor are mapped to formal representations in ΩMEGA. Does this mapping already imply that the informal proofs are validated? Clearly not, since ΩMEGA proof sketches at the TASK LAYER may be unsound and only full expansion of these proof sketches to the CORE-calculus layer will assure soundness. In our approach, this expansion can ideally be automated by ΩMEGA's reasoning tools. However, this clearly depends on the structural quality and the granularity of the informal proof. And, of course, if the informal proof is wrong, the expansion will fail and an interaction with the user to patch the proof is required.

References

[1] Marianthi Alexoudi, Claus Zinn, and Alan Bundy. English summaries of mathematical proofs. In Christoph Benzmüller and Wolfgang Windsteiger, editors, *Second International Joint Conference on Automated Reasoning — Workshop on Computer-Supported Mathematical Theory Development*, pages 49–60, University College Cork, Cork, Ireland, 2004.

[2] Ahmed Amerkad, Yves Bertot, and Laurence Rideau. Mathematics and proof presentation in Pcoq. In Uwe Egly, Armin Fiedler, Helmut Horacek, and Stephan Schmitt, editors, *Proceedings of the Workshop on Proof Transformation, Proof Presentations and Complexity of Proofs (PTP-01)*, pages 51–60. Universitá degli studi di Siena, 2001.

[3] David Aspinall, Christoph L¨th, and Burkhart Wolff. Assisted proof document authoring. In Michael Kohlhase, editor, *Mathematical Knowledge Management MKM 2005*, volume 3863 of *Lecture Notes in Artificial Intelligence*, pages 65– 80. Springer, 2006.

[4] S. Autexier. The CORE calculus. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, july 2005. Springer.

[5] Serge Autexier, Christoph Benzmüller, Dominik Dietrich, Andreas Meier, and Claus-Peter Wirth. A generic modular data structure for proof attempts alternating on ideas and granularity. In Michael Kohlhase, editor, *Proceedings of MKM'05*, volume 3863 of *LNAI*, IUB Bremen, Germany, january 2006. Springer.

[6] Serge Autexier and Dominik Dietrich. Synthesizing proof planning methods and oants agents from mathematical knowledge. In Jon Borwein and Bill Farmer, editors, *Proceedings of MKM'06*. Springer, august 2006.

[7] Serge Autexier and Armin Fiedler. Textbook proofs meet formal logic - the problem of underspecification and granularity. In Michael Kohlhase, editor, *Proceedings of MKM'05*, volume 3863 of *LNAI*, IUB Bremen, Germany, january 2006. Springer.

[8] Serge Autexier and Dieter Hutter. Formal software development in maya. In Dieter Hutter and Werner Stephan, editors, *Festschrift in Honor of J. Siekmann*, volume 2605 of *LNAI*. Springer, february 2005.

[9] Chr. Benzmüller and V. Sorge. ΩANTs – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Proceedings of Calculemus-2000*, St. Andrews, UK, 6–7 August 2001. AK Peters.

[10] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon. Xml path language (xpath) 2.0, 2005. http://www.w3.org/TR/xpath20.

[11] T. Bray, J. Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible markup language (xml) 1.0 (third edition), 2004. http://www.w3.org/TR/REC-xml.

[12] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. An Overview of the Theorema Project. In *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'97)*, Hawaii, 1997. ACM Press.

[13] Nicolaas G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In *Symposium on Automatic Demonstration*, pages 29–61, 1970.

[14] Dominik Dietrich. The Task Layer of the ΩMEGA System. Diploma thesis, Saarland University, Saarbrücken, Germany, 2006.

[15] Armin Fiedler. *User-adaptive Proof Explanation*. Phd thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany, 2001.

[16] Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of high-level formal proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and Eleventh Innovative Application of Artificial Intelligence Conference (IAAI-99)*, pages 277–284. AAAI Press, 1999.

[17] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. Number 112 in DISKI. Infix, Sankt Augustin, Germany, 1996.

[18] F. Kamareddine, M. Maarek, and J. B. Wells. Toward an object-oriented structure for mathematical text. In M. Kohlhase, editor, *MKM 2005, Fourth International Conference on Mathematical Knowledge Management*, LNAI 3863, pages 217–233. Springer, 2006.

[19] Michael Kohlhase. OMDOC: Towards an Internet standard for the administration, distribution, and teaching of mathematical knowledge. In John A. Campbell and Eugenio Roanes-Lozano, editors, *Proceedings of Artificial intelligence and symbolic computation (AISC-00)*, volume 1930 of *LNCS*. Springer, 2001. http://www.mathweb.org/omdoc.

[20] Andreas Laux and Lars Martin. Xupdate xml update language (working draft), 2000. http://xmldb-org.sourceforge.net/xupdate.

[21] A. Meier and E. Melis. MULTI: A multi-strategy proof-planner. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, july 2005. Springer.

[22] Svetlana Radzevich. Semantic-based diff, patch and merge for xml-documents. Master thesis, Saarland University, Saarbrücken, Germany, 2006.

[23] Aarne Ranta. Grammatical framework — a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189, 2004.

[24] Jörg Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, and Martin Pollet. Proof development with OMEGA: $\sqrt{2}$ is irrational. In Matthias Baaz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002*, number 2514 in LNAI, pages 367–387. Springer, 2002.

[25] 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. LΩUI: Lovely Ωmega user interface. *Formal Aspects of Computing*, 11:326–342, 1999.

[26] V. Sorge. *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning.* PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, Novermber 2001.

[27] Joris van der Hoeven. Gnu T_EX_{MACS}: A free, structured, wysiwyg and technical text editor. Number 39-40 in Cahiers GUTenberg, May 2001.

[28] Marc Wagner. Mediation between text-editors and proof assistance systems. Diploma thesis, Saarland University, Saarbrücken, Germany, 2006.

[29] Markus Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics: TPHOLs'99*, volume 1690 of *LNCS*, pages 167–184. Springer Verlag, 1999.

[30] F. Wiedijk. Formal proof sketches. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs: Third International Workshop, TYPES 2003*, LNCS 3085, pages 378–393, Torino, Italy, 2004. Springer.

[31] Dave Winer. Xml-rpc specification, 1999. http://www.xmlrpc.com/spec.

Appendix

A　PlatΩ's Interfaces

In this section we provide detailed descriptions of PLATΩ's abstract interface functions:

- **Initialize a session**:

      plato:init (client.name) -> session.name

  initializes a new session. It takes the client name (string) as only argument and returns the session name (string) or an error message. The purpose is to start a session in PLATΩ and in the proof assistance system in order to get a session identifier which can be used to indicate the working session in all following interactions with PLATΩ. This is important, for example, if the text editor user wants to get support for two or more documents, or if PLATΩ is launched as mathematical service provider to allow text-editors the access over the web.

- **Upload a document**:

      plato:upload (session.name, document) -> OK

  uploads a whole document in the informal language PL. The arguments are the session name (string), received previously by `plato:init`, and the document (string). It returns a simple OK (boolean) or an error message. PLATΩ first verifies the syntax of the document and then automatically builds up the corresponding formal representations DL and TL, which are uploaded into the proof assistance system. If a document has already been uploaded, PLATΩ performs an internal difference analysis using a semantic based differencing mechanism [22] and then proceeds as with patching the document.

- **Patch a document**:

      plato:patch (session.name, diff) -> OK

  patches an already uploaded document in the informal language PL with patch information. The arguments are the session name (string) and the patch information (XUPDATE, see Section 3.2). PLATΩ returns a simple OK (boolean) or

an error message. PLATΩ transforms this patch information into patches for the formal representations DL and TL, which are used to patch the datastructure of the proof assistance system.

- **Request a menu**:

```
plato:service (session.name, object.id) ->  menu
```

requests a menu for an object in the informal language PL inside the document. The arguments are the session name (string) and the unique identifier of the selected object (string). The response is either a menu in the service language SL (string) or an error message. The purpose is to use `plato:service` in order to get a service menu from the proof assistance system with actions for the selected object in the document. PLATΩ looks into his maptable for the corresponding objects in the formal representation and requests service support from the proof assistance system on these objects.

- **Execute a menu action**:

```
plato:execute (session.name, action.id, arguments)
                    -> (menu.diff, document.diff, custom)
```

triggers the execution of an action with its evaluated arguments. The arguments are the session name (string), the unique identifier of the selected action (string) and the arguments as a list of pairs with name (string) and value (string). It returns a list with a patch for the current menu (string), a patch for the document (string) and a custom answer (string), or an error message.

- **Close a session**:

```
plato:close (session.name) ->  OK
```

closes a session. The argument is the session name (string). It returns a simple `OK` (boolean) or an error message. The purpose is to terminate a session appropriately, such that PLATΩ as well as the proof assistance system are able to delete any information regarding this session.