Proposal for an action of technological development

Hugo Herbelin

6 avril 2008

1 Summary

This proposal is supported by the two INRIA teams Typical (Saclay - Île-de-France) and Marelle (Sophia-Antipolis) and the non-INRIA site CNAM. The project is coordinated by Hugo Herbelin and it requests funding for three engineers, and support for both visits and meetings between the development sites.

2 Introduction

The Coq system was designed with two complementary objectives in mind. On the one hand, it was designed as a tool to describe algorithms in such a way that the implementation of these algorithms could be *correct-by-construction*. On the other hand, it was designed as a tool to describe mathematical proofs in such a way that all details could *be verified by a computer*. The two objectives are actually intertwined because a lot of mathematical objects can be described as algorithms and the correctness of computer science algorithms may rely on mathematical proofs of arbitrary difficulty.

Recently, the system has attracted a wide user community, interested both in its power to describe advanced mathematics and in its capability to support reliable software development. Many scientific publications describe developments made using Coq, major conference organize tutorials on this system and its applicability to their field of interest, world-wide projects declare this system as their system of choice, Formal methods tools are designed to produce input for Coq, and some researchers even start to consider the Coq system as a viable "programming language and methodology" for most of their developments. INRIA is receiving a lot of notoriety from this tool, which plays an important role in transferring results from its research teams to the rest of the academic world and the industrial community.

The Coq development team has traditionally been composed of researchers with very little input by engineers. There are about 15 contributors this year, there have been 40 contributors since the implementation started. Recent developers are spread in various academic teams, mostly at INRIA Saclay and Sophia-Antipolis but also at the University of Paris 7, CNAM, and the University of Nijmegen. The architecture of the system is well-suited for collaborative work as it relies on a central kernel that can be maintained by a reduced set of core developers, while other developers can contribute external components whose failure does not endanger the reliability of the whole system[1].

[1] The current permanent core developers are Bruno Barras (TypiCal), Benjamin Grégoire (Everest/Marelle), Hugo Herbelin (TypiCal) and Pierre Letouzey (Paris 7), assisted by Jean-Marc Notin (TypiCal, CNRS) as a research engineer while Christine Paulin (ProVal) and Jean-Christophe Filliâtre (ProVal) keep a high expertise from previous involvement in the core development. The main non permanent persons involved are Pierre Corbineau (Nijmegen), Élie Soubiran (TypiCal), Matthieu Sozeau (ProVal) and Arnaud Spiwack (TypiCal). The main current contributors to peripheral components are Julien Forest (CNAM) and Pierre Courtieu (CNAM), with an involvement too from Yves Bertot (Marelle), Laurent Théry (Marelle) and Russell O'Connor (Nijmegen).

The purpose of this proposal is to boost the momentum of the Coq development team by improving its capability to work together and by adding manpower on technical aspects.

Three aspects have been isolated which have a clear impact on the usability of the system :

– ease of use for users of all levels of expertise : the objective is to ensure that beginners will find an integrated environment to work with Coq that is easy to install and will provide complete support to them as their level of expertise grows, up to configurability and extendability for researchers working in specific domains. We believe this aspect plays a significant role in acquiring a stable community of users : beginners need to be pampered in the first steps, or they risk to be attracted by competing tools. Once a user has chosen a specific proof system, it is very difficult to make her change to another one, because using a proof system imposes a strong learning curve, independently of user-interface problems.
– Inter-process communication for integration in tool chains : the objective is to cater for users who want to integrate Coq as a participating tool in larger projects. A commonly found approach is that third-party research teams develop tools for reliable software development and delegate part of their proving process to the Coq system ; other teams just want to use Coq on specific domains and develop specialized reasoning tools to which Coq can delegate some of the proof search tasks.
– Library management tools : part of the success of Coq is that libraries are already provided for most of the common mathematical and algorithmic knowledge, thus making it possible for newcomers to address directly interesting problems without needing to start everything from first principles. There is already an established tradition in the user-community to share libraries through a repository that is managed by the Coq development team. However, as libraries tend to grow, it is more and more important to ensure that users can easily find the right elements of the library for their needs.

We believe that these three technical aspects require manpower in the form of engineers with reasonable knowledge in programming (mostly functional programming in Objective Caml, since this is the language in which Coq is developed).

3 Context and state of the art

3.1 Technological and scientific context

Coq is the only (real-sized) proof assistant developed at INRIA. It is one of the 8 most used proof assistants in the world. In Europe, its main challengers are Isabelle (developed in Munich, Germany), HOL (developed in Cambridge, UK) and Mizar (developed in Białystok, Poland).

Coq is used in various research contexts and in a few industrial contexts. It is used in the context of formal mathematics at the university of Nijmegen (constructive algebra and analysis), INRIA Sophia-Antipolis (number theory and algebra), INRIA-MSR joint lab (group theory), the university of Nice (algebra). It is used in France in the context of computer science at INRIA-Rocquencourt (certified compilation), INRIA-Saclay (certification of imperative programs), LORIA. Outside France, it is used in the context of computer science e.g. at U. Penn (formalization of programming languages semantics), Yale, Ottawa and Berkeley Universities (building of a certified platform for proof-carrying code), University of Princeton (certified compilation), AIST at Tokyo (certification of cryptographic protocols), Microsoft Research Cambridge (proof of imperative programs). In the industry, it is used by Gemalto and Trusted Logic (JavaCard formal model and commercial applets certification).

All in all, it is difficult to evaluate how much Coq is used. Two indicators are the readership of the book on Coq and the number of subscribers to the Coq-club mailing list : as of February 2008, more than a thousand copies of the book have been sold and there are a little less than 600 subscribers on the mailing list. Coq is taught or used for teaching in many universities : Paris, Bordeaux, Lyon, Nice, Strasbourg, CNAM, Ottawa, U. Penn, Warsaw, Princeton, Yale, Berkeley, Rosario (Argentina), ...

3.2 Objectives

The objectives of the action is to increase the ability of Coq to communicate with other tools (package 1), to provide a good useful integrated graphical interface (package 2), to improve the visibility and reusability of the Coq user contributions (package 3), and to increase the ability for the Coq developers to visit and meet the users and the other developers (package 4).

3.3 Technological and economic constraints

We do not see any particular technological or economic constraints except perhaps that it may be difficult to find good programmers knowing that industry is often ready to spend more resources to attract the good programmers than academy does.

3.4 Positioning

4 Participants to the action

The participants to the action of technological development, listed by site, are the following :
- INRIA teams
  - TypiCal (ex-LogiCal)
    Participants to the action : H. Herbelin, B. Barras, P.-Y. Strub
  - Marelle (ex-Lemme, ex-Croap)
    Participants to the action : Y. Bertot
- Non-INRIA teams
  - CNAM
    Participants to the action : P. Courtieu, O. Pons

NOTE : There is a ongoing project aiming at creating a new INRIA team at University Paris 7 around Hugo Herbelin (from TypiCal), Pierre Letouzey (from PPS, Paris 7) and Pierre-Louis Curien (from PPS, CNRS). If the project receives a positive agreement, Hugo Herbelin would then be a member of a team depending on INRIA Rocquencourt-Paris.

5 Description of activities

5.1 Planning

We give a planning over the next two years.

n.	Deliverable Title	Partic.	Estim. person-month	Deliv. month
1.1	XML DTD of Coq main components	Mar	2	2
1.2	Coq support for XML input	Mar	4	4
1.3	Richly annotated pretty-printer	Mar	6	8
1.4	API for communicating with Coq	Mar	2	10
1.5	Specification of printer's tuning parameters	Mar	4	14
1.6	Tunable pretty printer	Mar	4	18
1.7	Implement. of the communication interface	Mar	4	22
2.1	Core integrated graphical interface	Typ/CNAM	6	6
2.2	Editing features	Typ/CNAM	2	8
2.3	Contextual menu features	Typ/CNAM	8	16
2.4	Views	Typ/CNAM	2	18
2.5	Proof-by-pointing features	Typ/CNAM	6	24
3.1	Metadata for user contributions	Typ	3	3
3.2	Web server for user contributions	Typ	2	5
3.3	Web server for Coq tools	Typ	3	8
3.4	Coq-Whelp communication implementation	Typ	4	12
3.5	Use of high-level output in rendering	Typ	3	15
3.6	Ergonomic issues of the server	Typ	3	18
4.1	Multi-site developers meeting	Typ	–	3, 6, 9, 15, 18, 21
4.2	User meetings	Typ	–	12, 24

Typ = TypiCal, Mar = Marelle

5.2 Deliverables

5.2.1 Structured communication between Coq and external tools

Participants : INRIA Marelle
Supervisor : Yves Bertot

In the current situation, developers of external tools can use Coq mostly by communicating through the interface that was designed for man-machine interaction. Limited support is also provided to let Coq proof procedures (known as *tactics*) call external proving tools, with a printing and parsing layers based on XML to pass data to and from the external tools. Communicating through the man oriented interface is made more difficult by the extensible nature of the parsing and pretty-printing layers of the Coq system. The objective of the work described in this section is to augment the communication capabilities with external tools with Coq so that a wide array of communication scenarios can be supported. This improvement will make Coq more easily integrated in a wide variety of research project, such as environments for formal mathematics, programming logic tools, domain specific languages, etc.

In general, Coq makes a lot of efforts to make its input and output concise and easy to read. Some of the capabilities are only concerned with parsing technology, other are only concerned with typing disciplines, while other handle a two-level interaction between the parser and the typing system. All these techniques have to respect the strong constraint of re-entrance, where the text output by Coq should be usable as input, for instance through copy-pasting operations.

Long term objectives We wish to improve on this situation in three directions :

1. XML Document Type Definitions (W3C's answer for this kind of need, henceforth abbreviated as DTDs) should be provided for the whole standard language of Coq, so that Coq documents could

be stored on disk in XML format as well as in the current syntax. This imposes that more XML DTDs need to be written.

2. The current pretty-printing facility of the Coq system should be extended to display structure information in the form of XML tags interspersed in the text. Thus, the same text could be used both for machine-processing (by ignoring some of the text content and relying on the structure information provided by the tags) or for human-oriented display (by hiding the tags and the structure information). We believe this extension should support "tunable" structure information : depending on some parameters, the extra structure information may or may not include information of some particular kind : coercions may be displayed or not, implicit arguments, may be displayed or not, structure information may be displayed only until a certain depth, or only for some sub-term. In particular, it should be possible to refer to sub-terms of previous output, so that this "text-based" facility could support the operations that are commonly done by pointing at a fragment of data.
3. A wider application programmer's interface (API) should be provided : currently XML-based communication is only provided in the `external` capability of the tactic language, whose purpose is to call an external tool. In this situation, Coq can only give a XML description of the current goal and the answer can either be a term of the logical language (the calculus of constructions) or a tactic name, applied to a collection of terms. We wish to make it possible to call external tools as part of other defining commands, or even as part of commands that do not change the state of the theorem prover. We also wish to make the theorem prover able to answer a wider variety of questions, such as parsing, or pretty-printing, type-checking, type-inference, unification, dependency, context searching, etc.

Priorities At the time of writing, we are aware that this set of objectives maybe too ambitious for a single engineer, however it is useful to keep them in mind. For this reason, this proposal contains the whole set but we propose to establish priorities :

– The requirement of more complete DTDs should be worked on first, because it should be a easy way to make a new developer aware of the extent of Coq's capabilities and input-output languages. However, we suggest that this task should be stopped after six months, so that only maintenance and debugging tasks should be kept until the end of the working period,
– The requirement of a tunable extension to the pretty-printing layer of Coq is a difficult task, but also one that can provide the strongest benefit, both for this task and for the next task (integrated graphical interface). It is difficult because it requires some reverse-engineering on the current pretty-printing mechanism. This pretty-printing mechanism is itself quite complex due to the extensible nature of the pretty-printing machinery. So we believe that this task should be started as soon the engineer has enough understanding of the context, and no later than the sixth month of the working period,
– The tunable nature of the pretty-printer extension makes that an arbitrary amount of energy can be spent in this domain. To avoid depriving the third point of attention, we suggest to interleave sub-tasks on the second point and the third point, possibly implementing entries of the application programmer's interface as soon as the extension to the pretty-printing mechanism makes them possible.

Identifying the risks We see three main areas of concern : overhead with respect to the evolution of Coq, difficulty of reverse-engineering Coq's elaborate pretty-printing machinery, and, in the long run, risk of quick obsolescence. The first, *overhead* comes from the fact that users regularly invent new approaches to communicating with the theorem prover, and these approaches get included in mainstream usage regularly. However, a user that invents a new command is often not competent enough to update the DTD and often they do not perceive the need for making sure the DTD follows the language evolution, because they only view Coq as a human-oriented processing tool. For this reason, the burden of making sure that the DTD follows the evolution of the language is a long term maintenance task that should be kept alive all through the working period. The second, *difficulty of reverse-engineering* is just a plain risk : we may not be able to hire an engineer that will be able to understand the pretty-printing machinery. This risk can be lessened by guaranteeing that the engineer will be in frequent interaction with researchers who have more knowledge of the current state of the machinery, but we should avoid a situation where researchers have to do the work in the engineer's place. The third, *obsolescence* is strongly related to the first one : if the machine-to-machine interface is not used enough, maintenance operations to follow the evolution of the Coq system will tend to be delayed and the interface will die out. For this reason, it is important that the gains of extending the structure-based communication of Coq should be illustrated by new commands that make the benefits visible to the widest possible community.

To preserve the compatibility between the last extension of the user language of Coq and a former version of it that a tool communicating with Coq would support, we hope that the use of XML attributes will help : extensions will be specified as much as possible under the form of new attributes so that non up-to-date software may still understand the structure of the exported Coq expression.

Work plan The work plan was already outlined in the section on priorities. Time units are given in months.

1.1 We expect the engineer working on this task to spend the first 2 months on getting acquainted with the Coq system, drafting a XML DTD for the main components of the standard language : Defining commands, tactics (the usual term for proving commands), term description language, query commands (task should be completed at T0 + 2).

1.2 Extend Coq so that it can receive its input in XML format. This implies finding a XML parsing package whose license conditions are compatible with Coq's and linking each XML construct to the corresponding Coq command or parsing capability (task should be completed at T0 + 4).

1.3 Study the pretty-printer of Coq and modify it so that it can add tags indicating the beginning and end of text corresponding to each sub-structure of an expression, together with a tag indicating the nature of the sub-structure (task should be completed at T0 + 8). We expect this task to require intensive support from researchers who have a good knowledge of Coq's input-output layers (mostly Hugo Herbelin, Yves Bertot also has some experience in this aspect).

1.4 Draft a specification of an application programmer's interface : give commands, informal meaning, type and format of inputs, type and format of output, effects on the state of the Coq system and its communication layer[2] (task should be complete at T0+10). *including possible delays in installation and understanding the problems, this could be the point reached at the end of the engineer's first year. The schedule starts again à T0+12*

1.5 Draft a specification of tuning parameters for the pretty-printing layer. There are several "dimensions" in which the pretty-printing tool can be parametrized, among which we can already cite : implicit arguments, coercions, depth, paths where detail should be displayed. The specification should study how these dimensions interact (does it make sense to display implicit arguments but not coercions, does it correspond to a need ?—task should be completed at T0+14).

1.6 Implement the tunable pretty-printer (task should be completed at T0+18).

1.7 Implement the application programmer's interface, together with a complete test suite for each of the primitives being implemented (task should be completed at T0+22).

The last two task, which contain code development, are given a larger time-span. We expect that these task will be slowed down by any mistake made in the previous tasks, which may have to be re-visited regularly.

[2]In particular, we envision commands that take paths as argument, which should be interpreted in the context of the result from a previous command : this kind of communication implies some notion of state.

5.2.2 Integrated graphical interface

Participants : INRIA TypiCal + CNAM
Supervisors : Hugo Herbelin (TypiCal), Pierre-Yves Strub (TypiCal), Pierre Courtieu (CNAM)

This deliverable covers the development of an integrated development interface for Coq written in Objective Caml, in the style of the existing CoqIDE but using the gtksourceview widget of GTK+ and offering new features not already provided by the currently used interfaces (mainly : searching tools, views, contextual menus, proof-by-pointing features).

Coq can currently be used through various interfaces :
- its integrated textual interface together with a line editor,
- its integrated graphical interface CoqIDE written in Objective Caml and LablGTK,
- the Emacs/XEmacs based Coq-specific instance of the generic Proof General interface written in Emacs Lisp,
- the Coq-specific instance of the ProverEditor Eclipse plug-in written in Java,
- the on-line web interface ProofWeb written in JavaScript.
- the no-longer supported Pcoq graphical interface written in Java,
- the abandoned TeXmacs-based interface.

In practice, the most used interfaces are Proof General and CoqIDE. The Eclipse plug-in, which has the minimal features so as to be usable, is rather slow and actually also misses advertising.

The experience shows the following :
- Objective Caml and its GTK+ binding allows for quick and easy development of directly usable new features. The current interface has 8 000 fresh lines of Objective Caml and offer basically the same features as Proof General (16 000 generic lines and 4 000 Coq-specific lines of Lisp) and rather more than the Eclipse plug-in (6 000 generic lines and 2 000 Coq-specific lines of Java).
- GTK+ provides with an elegant graphical rendering and it is supported without any particular installation on the three most common architectures (on MacOS X, it requires the installation of the X11 packages).
- Coq developers easily extended CoqIDE with new features according to their specific need (see e.g. J. Chrząszcz with his Papuk extension, P. Corbineau for his declarative proof language). The use of Objective Caml, in addition to providing quick prototyping, contributes to easing the joint extension of new features in Coq with new features in the interface. We strongly believe that the tight connection between Coq and its interface helps at innovating, while generic interfaces such as Proof General or Eclipse are more inclined to provide "universal" features.
- CoqIDE has contributed to stimulate the evolution of the other graphical interfaces : for instance in-place error localization is a feature introduced by CoqIDE.
- Compared to Emacs Proof General, CoqIDE misses some useful Emacs-specific key bindings (such as Meta-B, Meta-F, ...) and it did not correctly implement search, replacement and word completion as efficiently as Emacs does. Because it is based on GTK+, CoqIDE will still provide basic bindings in either Emacs-style or Windows-style.
- CoqIDE is fragile on some versions of Windows due an excessive use of the GTK+ thread API (the API works differently in UNIX and Windows). This is a serious concern and a new CoqIDE gtksourceview-based core has been made by Pierre-Yves Strub to solve this problem.

Both Proof General and CoqIDE provide the minimal set of features for an interactive use of Coq, so why to want more, and why to put the effort on an interface written in Objective Caml and GTK+ ?
- New features, such as expressive search in the whole library using the results of package 3 (communication between Whelp and Coq) will typically have to be made available from the user interfaces. To ensure that this will be done in at least one interface, we believe that the best way is to do it in the interface we have the opportunity to code in, namely the interface released with Coq.

- How and where to implement many convenient features (such as the one listed in the Appendix) that are either not available in the current interfaces or that are not easily usable in existing interfaces ? Being able to directly implement our new ideas on a directly usable interface requires to have at our disposal an interface such as CoqIDE.
- Objective Caml is a good language for rapid prototyping.
- We believe that a graphical interface directly available at the installation of Coq without need for further installations is important for not discouraging fresh users. A bundled interface is the only possible choice here.
- Providing new features in a user interface does not mean that the non purely graphical part of the code is not made available for reuse from other graphical interfaces. Especially, the collaboration with P. Courtieu, who is the developer of the Coq part of the Proof General interface, will grant the reusability of the components.

Description of the tasks

2.1 Core integrated interface
This covers the communication structure and the graphical layout (edition, tactic and output windows, right-click structure, menus).
2.2 Editing features
This covers : incremental or global parsing, incremental or global backtracking, search, replacement, basic editing key bindings.
2.3 Specific contextual features
2.4 Views
2.5 Proof-by-pointing
(see Appendix A for details – in French)

We expect this task requiring 2 years of work for a good engineer.

The decision process will be shared by the three supervisors. The engineer will be hosted at TypiCal where Hugo Herbelin and Pierre-Yves Strub are. Interaction with Pierre Courtieu will happen as often as necessary and at least twice a month.

We will have a questionnaire circulating at the end of the action to poll users on whether the new interface fulfills their need more than the other interfaces they know do.

5.2.3 Online browsing and seeking through the Coq user tools and contributions

Participants : INRIA TypiCal
Supervisors : Hugo Herbelin (TypiCal)

In the current situation, users can submit developments in Coq that are added on a server and maintained up-to-date with the evolutions of Coq. These contributions are simply added on the server, classified in a thematic index and downloadable as compressed archives. Users cannot upload and modify their contributions and visitors of the site cannot search or browse these contributions.

Objectives The objective is to improve on this situation by extending the repository of Coq developments into a repository also of Coq tools and into a repository of browsable, seekable, and updatable developments, together with dependency links between the different developments.

- A repository of Coq third-party's tools
  More and more third-party extensions of Coq are developed. For instance, Lande developed at IRISA an automation tool for arithmetics in Coq, Mathematical Components at the INRIA-Microsoft lab developed a language of tactics and an associated methodology for "small-scale

reflection", Berkeley developed an SMT solver on top of Coq, Marelle at INRIA Sophia developed a decision tactic for polynomial inequalities, the university of Warsaw developed a graphical tool dedicated to teaching. The experience of Objective Caml and of the related Caml Hump shows that the ability of setting up (and managing) a repository of Coq's extensions is valuable for the users and for the further development of the work on Coq.

– Valorisation of the existing developments : dependency constraints, online seeking and browsing
  For Coq productivity, being able to quickly access the standard libraries and/or the user contributions to find the name of the lemma that will solve the next goal is important. Currently, the practice for searching a lemma is often, unfortunately, to use a `grep` through a directory of files.
  The Matita team at the University of Bologna opened the road by setting up an hypertextual repository of formal mathematics on top of which they develop seeking, browsing and rendering tool (this was partly done in the context of the MoWGLI European project that Bologna initiated and to which Marelle, LogiCal and Nijmegen contributed).
  The seeking tool provided by the Matita team is remarkably powerful and efficient thanks to the design of clever linear-time searching algorithms. There is a default yet : the whole architecture is very complex and actually too complex for what is really needed.
  The browsing tool provided by the Matita team can be improved. Indeed, there are different ways to formalize a proof, different ways to tell the proof assistant how to perform the proof and the current rendering engine only knows one way. Moreover, rendering in natural language tends to obfuscate the structure of the proof by putting every kind of mathematical inference at the same level of detail. Our intuition is that the work on rendering cannot be disassociated from the work on formalizing in a mathematical-style language.
  Rendering is also performed through the use of stylesheets in the XSLT language which is a verbose and non robust language. The practice seems to have shown that a small modification of an XSLT transformation can unwantedly impact the correctness of other transformations without any support from the language to protect against these inopportune breakages. This suggests that for liveness and scalability of the transformations, the use of a more robust language like ML would be better.
  To ensure correct dependencies between the different contributions tools are missing. Similarly for the dependencies against the Coq version numbers.

Description of the tasks

3.1 Definition of a collection of metadata to be bound to individual user contributions
Such a collection of metadata already exists (it contains authors, Coq version, web page, description, etc). What are missing are e.g. some criteria of classification (using e.g. standard thematic classification scheme such as the ACM codes), more precise versioning system (which Coq versions it works with exactly), refereeing notices, ...

3.2 Set up of a web server with the following features :
- Two sections for : submitted contributions and freely uploadable part.
- Various ordering classifications of the user contributions (per site, per author, per thematic, per size, per date, etc).
- Hyper-textual rendering of the user contribution using a mix of coqdoc (the Coq documentation tool) and of the HELM-MoWGLI tools of Bologna (proof in natural language), with the possibility to switch between different representation of formulas and proofs (low-level expressions of the Calculus of Inductive Constructions, terms in natural language, proof scripts, high-level representations with notations and implicit contents, hiding of details).
- Computation and elegant display of the dependency graph between contributions.
- Automatic submission page.

3.3 Set up of a server for external Coq tools, in the spirit of the Caml Hump.
3.4 Implementation of the protocol to send local environment of Coq to the Bologna Whelp searching tool.
3.5 Implementation of a protocol to share implicit contents and notations with the rendering tool (in accordance with package 1).
3.6 Ergonomy issues with the proof rendering tool

We expect this task requiring about one year and a half work for a good engineer.

5.2.4 Meetings

Participants : All teams involved in the development of Coq
Supervisors : Hugo Herbelin and Bruno Barras (TypiCal)

We plan developer meetings at the European level 3 or 4 times a year. To start, we plan to have meetings on the following subjects :
4.1a A meeting on the standard library.
4.1b A meeting on graphical interfaces which would involve the following participants : Everest (Eclipse plug-in), Marelle (the Pcoq experiment), LogiCal/ProVal (CoqIDE), Nijmegen (ProofWeb), CNAM (Proof General), Nice (Coqweb).
4.1c A meeting on automation in Coq which would involve the following participants : LogiCal (resolution, decision on rings, real numbers, in arithmetics, communication with CAS, ...), ProVal (Ergo, communication with SMT solvers), Marelle and Everest (simplification on rings and fields, polynomial inequalities (Sum of Square)), Nijmegen (firstorder, congruence, the "math" mode) + external speakers : Damien Doligez (Zenon, MSR-INRIA), Adam Chlipala (Kettle, Berkeley).
4.1d A meeting on sharing needs and proposal for a "better" language of procedural tactics : LogiCal, Marelle, PPS, Nijmegen + external speakers : Gonthier (SSReflect).
4.1e A meeting to share the experiment and needs for arithmetical, algebraic and real numbers library that would influence the person(s) in charge of releasing such nice stable uniform libraries.
4.1f A meeting on the architecture of Coq and the documentation of its architecture which would involve all groups developing ML code for Coq and which could invite ML developers from groups abroad (e.g. the developers of Kettle in Necula's group).
4.1g We plan other meetings on other or on the same subjects depending on the need for them.
4.2 We plan a user meetings at the world level once a year, typically as a side event of a conference (probably TPHOLs).

5.2.5 Development mode and reporting

We will expect the programmers to write well-structured code with adequate documentation.
We will issue indicative reports on how the action is working every 6 months and a more detailed report at the conclusion of each package. In case of specific problems happening during the action, we may issue exceptional reports.

6 Resources

6.1 Human resources

– One 2-years engineer for the definition and implementation of the richly annotated pretty-printer and of the communication API of Coq.
  We expect a good programmer familiar with functional programming (ideally Objective Caml).

– One 2-years engineer for the development of a graphical interface.
We expect a good programmer familiar with functional programming (ideally Objective Caml), GTK+, edition widgets (ideally gtksourceview), and with a minimal background in ergonomy.
– One 1-year and a half engineer for setting the browsing and seeking platform.
We expect a good programmer familiar with functional programming (ideally Objective Caml), database management (preferably MySQL), web services installation and, taste and techniques in web design.

6.2 Missions

Each developer meeting would gather for one or two days a number of 16 people in the average, mainly from France, with funding for 4 persons from Sophia-Antipolis or Nijmegen when in Paris and from 11 persons from Paris or Nijmegen when in Sophia (we don't plan to have developer meetings in the Netherlands). One non European may be invited once a year (e.g. G. Necula from California or D. Nowak from Japan). Based on an approximative cost of 750 € for moving between sites, this makes 20000 € a year. See details on Figure 1.

Each yearly user meeting would gather for one or two days between 40 and 50 persons, from France, the Netherlands, Poland, the USA, Japan, ... There are two options, either a meeting in France with funding for users coming from abroad or a meeting organized jointly to a conference with funding for developers. The cost may vary a lot depending on the location of the meeting. Based on 8 missions to Japan or the US at an average cost of 2500 €, this makes 20000 € a year.

We would be happy too to use funding for a few one-week visits a year between partners (a number of three visits seems to be a reasonable upper bound). The estimated cost here is 1600 € for the journey and a five-day stay.

7 Miscellaneous remarks

Coq is released under the LGPL 2.1. The software realized in the context of the action is intended to be released under the same terms.

Diffusion of the information is intended to be through the Coq web site and the Coq mailing lists as it is commonly done for advertising new release and new tools.

Industrial transfer would be welcome if ever some partner thinks it is worth too but it is not an objective of the action.

Depending on the evolution of different ongoing projects at the European level (especially the possible setting of a Math Wiki), the platform of Coq contribution may itself evolve in view of a tight collaboration with these projects.

8 Evaluation

Here is a list of heavy Coq users that may help INRIA to evaluate the result of the action : Xavier Leroy (INRIA-Rocquencourt), Andrew Appel (Princeton University), Benjamin Pierce (Yale University), André Hirschowitz (Université de Nice).

Event	# of people from Paris in the average and cost	# of people from Sophia in the average and cost	# of people from Nijmegen in the average and cost	Other costs	Total cost per event	# of such event a year	Total cost per year	Total cost for 2 years
Devel. meeting in Paris area	12×0€ = 0€	3×750€ = 2250€	1×750€		3000€	3	9000€	18000€
Devel. meeting in Sophia	10×750€ = 7500€	5×0€ = 0€	1×1000€		8500€	1	8500€	17000€
Invitation from abroad				2500€[a]	2500€	1	2500€	5000€
User meeting abroad	6 developers $\times$ 2500€ (all sites together)			2×2500€ [b]	20000€	1	20000€	40000€
5-days visits within Europe				1600€[c]	1600€	3	4800€	9600€
Total								89600€

FIG. 1 – Budget for developer meetings, user meetings and visits

[a]one personality, intensive user or developer of an extension of Coq, from US or Japan

[b]invited speakers

[c]estimated cost of the visit

Annexe

A Discussion et objectifs détaillés concernant le projet d'interface graphique intégré

A.1 Situation actuelle

Nous donnons ci-après une description de la situation actuelle concernant les interfaces graphiques existant pour Coq.

A.1.1 Proof General

Proof General est une interface pour Coq basée sur Emacs. Elle permet le développement pas à pas de sessions Coq. La fenêtre Emacs est divisée en deux parties.

- Dans la première partie de l'écran, l'utilisateur valide les lignes de son développement, une à une, ou un bloc à la fois. Les parties validées sont grisées et ne peuvent plus être modifiées. Les parties non validées restent modifiables. Un mécanisme permet d'annuler une ligne validée et la partie annulée redevient modifiable.
- Dans la second partie de l'écran s'affichent les réponses de Coq, ou, dans le cas de la mise en œuvre d'une preuve interactive, la liste des formules restant à prouver.

En plus de ce mécanisme de développement interactif, Proof General fournit une collection de fonctionnalités, les principales d'entre elles étant :

- colorisation des mots-clés ;
- différents modèles de définition pré-enregistrés ;
- items de menu redirigeant vers des fonctions Coq (telles que recherche dans l'environnement, ou affichage de la valeur d'une constante).

Du point de vue de l'implantation, l'interface Coq Proof General se décompose en deux parties. La première partie (16000 lignes de code Emacs) est générique et indépendante de tel ou tel assistant de preuve. La seconde partie (4000 lignes) connecte la partie générique à la partie spécifique Coq. Le développement de Proof General a commencé en 1994 mais l'interface n'est devenue générique, et n'a adopté le nom Proof General, qu'à partir de 1998/1999.

La partie générique de Proof General est essentiellement développée par David Aspinall à l'université d'Édimbourg. La partie spécifique Coq fut essentiellement développée par Pierre Courtieu au CNAM.

A.1.2 CoqIDE

CoqIDE (pour Coq Integrated Development Environment) est une interface graphique pour Coq dévelopée pour l'essentiel par Benjamin Monate dans l'équipe ProVal en 2003. Elle est écrite en Objective CAML et repose sur GTK+ pour les fonctionnalités graphiques et pour la communication interne avec Coq (utilisation de « threads »).

Sa facilité d'utilisation l'a vite rendue populaire. L'interface innovait en particulier par un environnement graphique très intuitif, qui, contrairement aux interfaces Pcoq et Proof General de cette même période, ne nécessitait pas la lecture d'un manuel pour être prise en charge.

CoqIDE a eu un rôle stimulant sur le développement de Proof General qui a alors adopté à son tour le même type de boutons intuitifs permettant d'avancer ou de reculer dans une session Coq. Proof General a aussi adopté la localisation des erreurs de CoqIDE, résultant en de grands gains de temps pour l'utilisateur final en cas d'erreur de sa part.

A.1.3 Comparaison entre CoqIDE et Proof General

À l'heure actuelle, CoqIDE et Proof General offrent sensiblement les mêmes fonctionnalités et les différences sont maintenant plus d'ordre architectural :

- CoqIDE est intégré à Coq et ne nécessite donc aucune installation particulière de la part de l'utilisateur.
- CoqIDE est parfois instable et conséquemment inutilisable sous certaines configurations de Windows.
- CoqIDE étant lié avec Coq, tout échec du processus Coq (ce qui peut arriver avec Objective Caml en présence de dépassement de la capacité de la pile) détruit le processus CoqIDE en même temps.
- CoqIDE n'offre pas la même richesse de « key bindings » que Emacs ce qui est souvent frustant de la part d'utilisateurs habitués à Emacs (seuls les raccourcis de base tels que Ctrl-A, Ctrl-E, Ctrl-F, Ctrl-B, Ctrl-/ sont fournis)
- Coq Emacs Proof General n'est pas particulièrement plus intéressant qu'autre chose pour les personnes non habituées à Emacs.
- Coq Emacs Proof General peut être étendu sans connaissance particulière du source de Coq pour quelqu'un habitué à programmer en Emacs Lisp. Par exemple, la gestion de la couleur des mot-clés ou de l'indentation peut être modifiée par de tels experts Emacs.

A.1.4 Proof General, PGIP et Eclipse

Les développeurs de Proof General envisagent de progressivement recommander l'utilisation de Proof General au dessus de Eclipse plutôt qu'au dessus de Emacs. Nous imaginons que les gains espérés pour l'utilisateur final sont les suivants :

- Eclipse est (apparemment) relativement populaire dans certains milieux, notamment dans certains milieux industriels ou dans certaines écoles d'ingénieurs.
- Eclipse offre en standard des fonctionnalités de gestion de projets : accès facile à des répertoires de fichiers faisant partie d'un même projet, fonctionnalités permettant la compilation facile de ces projets, recherche rapide de fonctions définies dans les fichiers du projet.

Notons qu'un plugin Eclipse pour Coq (plugin indépendant de Proof General) a été développé par Julien Charles à l'INRIA Sophia-Antipolis. Ce plugin fournit les fonctionnalités de base de Coq Emacs Proof General et CoqIDE (validation incrémentale des preuves et mécanisme d'annulation).

A.1.5 Pcoq

Pcoq (anciennement CtCoq) est une interface ambitieuse capable, contrairement à Proof General et CoqIDE, d'édition structurée : la sélection d'un caractère à la souris identifie automatiquement la plus petite sous-expression contenant ce caractère. Pcoq fut innovant aussi dans ses fonctionnalités de proof-by-pointing et de rendu des preuves en langue naturelle.

Pcoq fut développé par Yves Bertot à l'INRIA Sophia-Antipolis. Le développement en est suspendu depuis 2003 et la dernière version n'est plus compatible avec la version courante de Coq.

A.1.6 Autres interfaces graphiques

ProofWeb est un client Javascript permettant d'utiliser Coq en ligne derrière un serveur localisé à l'université de Nimègue. ProofWeb propose des fonctionnalités interactives de base comparables à celles de CoqIDE et Proof General avec une spécialisation vers l'enseignement de la logique. Il semble irréaliste d'utiliser Coq en ligne pour des applications autres que éducatives ou collaboratives (wiki par exemple).

Papuk est une extension de CoqIDE dédiée à l'utilisation de Coq pour l'enseignement de la logique.

Coq-TeXmacs est une interface de Coq pour la rédaction de documents TeX contenant des scripts de preuve.

A.2 Situation envisagée

Dans la pratique, au delà des fonctionnalités de base fournies par les différentes interfaces, nous croyons que la prochaine étape en terme de valeur ajoutée potentiellement fournie par une interface graphique se situe dans

- la qualité des outils de recherche accessible au travers de l'interface (recherche multi-critère sur une base de recherche large) ;
- la capacité à passer d'une présentation à l'autre d'un document ou d'une expression ;
- les fonctionnalités contextuelles spécifiques associables à des expressions, sous-expressions ou identificateurs ;
- la preuve graphique par clic, par menu contextuel, ou par glisser-déposer (proof-by-pointing) particulièrement intéressante pour les débutants ou pour la manipulation d'expressions complexes.

À notre connaissance, ce genre de fonctionnalités n'est vraiment implantée par aucune des interfaces graphiques (à l'exception de Pcoq qui n'est pas maintenu) et nous pensons que c'est la direction vers laquelle nous devons aller.

A.3 Paramètres du choix entre une interface générique et une interface intégrée

L'intérêt pour une interface graphique intégrée est

- sa facilité d'installation pour l'utilisateur ;
- la possibilité d'une adéquation idéale entre les fonctionnalités fournies et les fonctionnalités attendues, sans superflu ;
- la facilité à faire évoluer l'interface en même temps que l'assistant de preuve et à expérimenter de nouvelles fonctionnalités (comme p.ex. pour un langage en style mathématique tel que celui implanté en Coq par P. Corbineau).

À l'inverse, l'intérêt pour une interface graphique indépendante est

- de profiter a priori des efforts d'implantation faits par les développeurs spécialistes d'interface ;
- de stimuler l'ajout de fonctionnalité par des développeurs indépendants qui n'ont à connaître que le langage d'implantation de l'interface (Lisp ou Java) et les protocoles de communication structurée avec Coq, sans avoir à dépendre des sources de Coq et de son langage d'implantation (Objective Caml).

Dans la pratique, les nouvelles fonctionnalités des interfaces graphiques (notamment de Proof General) ont été suscitées par des nouveautés de CoqIDE (localisation des erreurs, boutons de navigation) et on peut légitimement se demander si le principe de laisser la possibilité à des développeurs indépendants de mettre en œuvre de nouvelles fonctionnalités constitue vraiment en soi une garantie que ces développeurs vont réellement innover avec de nouvelles fonctionnalités interactives.

Dans la pratique, les défauts de CoqIDE (instabilité sous Windows, solidarité fatale en cas de mort prématurée du processus Coq sous-jacent), sont résolus dans un nouveau prototype de CoqIDE initié par Pierre-Yves Strub de l'équipe TypiCal (instabilité résolue par l'utilisation de uniquement deux threads, solidarité avec Coq résolue par l'effort de Objective Caml pour ne plus mourir en cas de dépassement des capacités de la pile).

Les arguments suggérant que l'architecture d'une interface intégrée est trop centralisée pour permettre des extensions (par exemple autour de la colorisation) sont des bons arguments. Aussi, nous pensons qu'une interface intégrée doit en partie fournir des mécanismes d'extensions : les règles d'indentation ou de colorisation doivent notamment être associées à des fichiers configurables, une possibilité de plugin doit pouvoir être envisagée.

A.4 Décision

Notre choix est de porter notre effort vers la reconstruction d'une interface graphique intégrée du type de CoqIDE, basée sur le prototype existant de Pierre-Yves Strub et sur le widget gtksourceview. Nous pensons que c'est la voie qui permet le plus d'innovations en terme d'interface, d'abord parce que le langage d'implantation sera celui des développeurs Coq qui seront facilement tenté d'étendre l'interface en fonction de leurs besoins comme plusieurs l'ont fait pour CoqIDE, ensuite parce que ce langage, Objective Caml, est un langage idéal pour la rapidité de prototypage.

A.5 Liste des fonctionnalités attendues

- Fonctionnalités contextuelles (bouton-droit sur expression du document) :
  - Obtention du type d'un identificateur (ou d'une expression close)
  - Obtention de la valeur d'un identificateur (commande Print)
  - Obtention de la liste des lemmes dépendant d'un identificateur
  - Liste des axiomes dont dépendent un identificateur
  - Tactiques applicables liés à cet identificateur
  - Explicitation du contenu implicite (Set Printing All) dans une sous-expression (difficile !)
  - Contenu extrait de l'objet en question

  Dans les cas d'un affichage, celui-ci doit se faire dans la fenêtre de sortie (sans forcément effacer la fenêtre au préalable) ou dans une division de la fenêtre de sortie.

  [Notons que dans le CoqIDE actuel, l'affiche du type d'un objet se fait en un double-clic (pour sélectionner l'identificateur, à supposer qu'il n'a pas de "_" dans son nom), puis F2, puis déplacement de la fenêtre pop-up dans un coin de l'écran pour la garder sous le coude]
- Fonctionnalités graphiques
  - Sélection d'un identificateur par double-clic (caractéres non alphanumériques, tels que "_" et " ' " inclus, contrairement au CoqIDE actuel)
  - Maximalisation des fenêtres par simple clic (fonctionnalité typique d'Eclipse)
  - Visualisation des répertoires « projet » sous forme de fenêtre (fonctionnalité à la Eclipse)
  - Empilement de l'historique des commandes récentes (à la Debian Aptitude, à la Moin Moin, etc)
  - Possibilité de replier certaines zones de la fenêtre, par exemple le corps d'une définition ou d'une preuve pour faciliter la vue générale d'un développement.
- Raccourcis
  - Bouton pour compiler (ou compilation automatique dès l'arrivée en fin de fichier),
  - Bouton pour appliquer coqdoc sur la fenêtre
- Modes
  - Bouton pour commuter l'affichage bas niveau (Set Printing All)
  - Support pour l'extension Papuk de Varsovie
- Fonctions de recherche
  - Recherche textuelle dans les fenêtres de la session (fonctionnalité Ctrl-s de Emacs).
  - Branchement de Coq sur Whelp avec gestion de l'environnement local.
  - Fenêtre de recherche avancée avec branchement sur la commande Coq SearchAbout.
  - Menu pour se rendre directement à une définition ou un lemme
- Fonctionnalités diverses
  - Colorisation des mots-clés paramétrable par fichier.
  - Sauvegarde automatique régulière
  - Unicode rendu correctement.
  - Complétion à la M-/ en Emacs.
  - Navigation de base avec, au choix, conventions à la Emacs (Ctrl-a, Ctrl-e, Ctrl-f, Ctrl-b, etc) ou

à la Windows.

- Preuve par glisser-déposer (proof-by-pointing)
  - Déplacement d'une hypothèse du but vers le contexte pour faire un "intros until - move".
  - Déplacement d'un sous-terme vers le but ou vers un sous-terme du but pour faire un "generalize".
  - Double-clic sur une hypothèse pour faire un "induction".
  - Possibilité de marquage des occurrences avant de faire une induction.
  - Boutons dans la marge pour les tactiques automatiques courantes.
  - Déplacement d'une hypothèse vers une autre pour faire soit un "apply in" soit un "rewrite in", en fonction de la forme de l'hypothèse déplacée
  - Double-clic sur un nom de constante pour le déplier

Questions ouvertes :

- Opportunité d'une reconnaissance des expressions... utile pour le proof-by-pointing.
- Opportunité de mettre en place un mécanisme de notations bidimensionnelles.