ED 427 654 IR 019 173

AUTHOR	Capani, Antonio; De Dominicis, Gabriel
TITLE	Web Algebra.
PUB DATE	1996-10-00
NOTE	9p.; In: WebNet 96 Conference Proceedings (San Francisco, CA, October 15-19, 1996); see IR 019 168. Pages contain light type.
AVAILABLE FROM	Web site: http://aace.virginia.edu/aace/conf/webnet/html/146.htm
PUB TYPE	Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	*Algebra; Computation; *Computer Interfaces; Computer Oriented Programs; Computer Software; Databases; Higher Education; *Man Machine Systems; Models; Navigation (Information Systems); *Online Systems; Questioning Techniques; Secondary Education; *World Wide Web
IDENTIFIERS	*Computer Algebra; Computer Assisted Mathematics Program; *Query Processing

ABSTRACT

This paper proposes a model for a general interface between people and Computer Algebra Systems (CAS). The main features in the CAS interface are data navigation and the possibility of accessing powerful remote machines. This model is based on the idea of session management, in which the main engine of the tool enables interactions with the Session Database. The model was implemented using the client-server capabilities of the World Wide Web. Possible applications of this model include educational purposes, computing, data navigation, communication of structured data, and simulation of operations on distributed systems. Topics discussed include the goal of the project; interface requirements; the session manager; sessions; applications, processes, and agents; queries; reports; scheduling of queries; crash recovery; and educational issues. (DLS)

WEB ALGEBRA

Antonio Capani, Department of Computer Science,
University of Genova, Italy, capani@disi.unige.it
Gabriel De Dominicis, Department of Mathematics,
University of Genova, Italy, dedo@dima.unige.it

Abstract: In this paper we propose a model for a general interface between people and Computer Algebra Systems (CAS).The main features in our CAS interface are data navigation and the possibility of accessing powerful remote machines. Our model is based on the idea of Session Management. We implemented our model using the client-server capabilities offered by WWW. We envision our proposal as being useful for: educational purposes; computing; data navigation; communication of structured data; simulation of operations on distributed systems.

1 Introduction

Our goal is the construction of a common *World Wide Web* interface for different *Computer Algebra Systems* (CAS for short) particularly for systems devoted to *Commutative Algebra computations*. We need to use different systems because each of them is powerful for doing particular tasks. We can mention, among non-commercial systems developed by academic research groups, CoCoA[CNR], *GB*, *Macaulay*[MA], *Singular*[Sing] and the *PoSSo* server. All of them are capable of performing basic arithmetic operations on algebraic objects like: multivariate polynomials; rational functions; polynomial matrices; ideals. At the same time, advanced operations like Grobner bases[BU][RO] of ideals and modules, elimination, computation of syzygies and resolution of both ideals and modules are supported in all these systems.In addition some of these systems are designed to perform specific algorithms like, for instance, *Hilbert function*, *Hilbert driven Buchberger* Algorithm[GMRT] or *Tangent Cone Algorithm*[TC] in a particularly efficient way. These systems are mainly used as research tools but some of them are also currently being used for teaching purposes.

Because of these considerations it is clear that a common way to interact with the previously mentioned systems would be advantageous.

A prototypal version of our interface has been already demonstrated during the Workshop "Calcolo Simbolico" (November 20-21, 1995 Genova Italy) and during the Conference "Algorithmic Algebraic Geometry and Singularity Theory" (Schloss Dagstuhl, Germany, Jan 22 - 26 1996).

1.1 Interface requirements

The starting point of our investigation was our experience both as users and as (co)-workers on the CAS system CoCoA. In our experience we noted that the interaction between users and systems is characterized by mixture of hard computations (e.g. operations that involve Grobner basis computation) and data inspection. Data inspection (or better *data navigation*) is a key point in CAS interfaces since we usually work with huge and complex data. So we have to consider the possibility of different *views* of the same output.

Since many computations are quite hard it is sometimes necessary to use suitable computational resources that are not locally available. So we were challenged by the possibility of adding *client-server capabilities* to the interface.

Another good feature would be, of course, *independence of architectures* (machine, operating system, graphical interface) at least from the client side of the interface.

We focused our attention on the possible use of a WWW interface. This choice gives us an unconditioned independence from the architecture (a user simply runs some browser), hypertextual data navigation, automatic client-server capabilites (due to the use of HTTP servers) and some other advantages like a complete abstraction with respect to the computational resources we are interacting with.

To meet our requirements, we have organized the interactions between users and applications in sets of queries called *sessions*.The collection of all information about sessions and applications is called the Session Database (SDB for short).

The SDB is updated by a server, called the Session Manager SM for short), via a set of services: creation of new sessions; submission of queries; execution of applications. Starting from the SDB, the user can create reports to organize the outputs of the queries.

The SDB is stored on a single machine. Each operation is performed by running a process on the machine of the SDB.

Its important to note that our interface is interactive, that is, each query submitted to an application process may depend on the previous queries submitted to the same process. Moreover we can start a computation, turn off our computer, and then, possibly after several days, reconnect to the session to see the results.

The WWW interface to the SM is called WebSM. The WebSM is connected to the SM via a set of *cgi-applications*, one for each service. Furthermore, with the WebSM the reports can be hypertextual.

For use the hardest point to investigate was related to *client-server* capabilities of the interface. If we plan to build such a type of interface, first of all we have to solve some specific problems like *crash recovery.*

We will distinguish between local-crash recovery (usually due to net problems or browser crash) and remote-crash recovery (roughly speaking, crash of the SM or failure of cgi-applications).

2 The Session Manager

The Session Manager (SM for short) is the main engine of our tool. It is a server that enables interactions with the Session Database (SDB). WebSM is a WWW interface to the SM. Here is a bit of terminology:

An *application* is an executable program; an *application process* is an application in execution; an *application process agent* is a process on the same machine of the SM which passes the queries of the SM to the application processes; a *query* is a tuple `<i,o,Status,Infos>`, where `i` and `o` are respectively the input and the output of the query, `Status` is the current status of the query (see Sect. 2.3) and `Infos` contains auxiliary information (not specified here) used by the SM; a *session* is a set of queries; a *Sessions Database* (SDB) is a tuple `<A,P,S>`, where `A` is a set containing the information related to the applications connected to the SM (i.e. the machine in which the application runs, the agent of the application, etc.), `P` is a set of application processes and `S` is a set of sessions.

The user interacts with the application processes by means of the SM which offers the following services: `NewSession()` creates a new session; `AddQuery(s,i)` adds the query `<i,Null,Input,Infos>` to the session `s`; `StartAppl(s,a)` starts the application `a` within the session `s` by running a new application process; `SubmitQuery(s,q,p)` enqueues the query `q` of the session `s` to the queries for the process `p`; `RestartProcess(s,p)` starts a new process `p'` for the same application of `p` and passes all the queries queued for `p` to the new process `p'` (used in case of an abnormal termination of `p`); `RecoverProcess(s,p)` is the same as `RestartProcess()`, except that it also submits the previously answered queries to the new process in order to restore the state of the process as it was at the moment of the crash; `Report(Params)` creates a report according to the specified parameters (see Sect 2.4).

The SM solves the queries using the application processes. The SM takes a query q submitted by the user to a process p and sends the input to the agent of p. When a process has completed a computation, the SM puts the result in the output part of the associated query.

In the following sections we will examine, in more detail, how the SM and the WebSM work. We will support our presentation with a simple example, which we now introduce.We start with an empty SDB, and we consider two different sessions s_1 and s_2. The two sessions use two different processes, p_1 and p_2, for the same application (CoCoA), but they share the process p_3 for running the application *singular*. We will present the status of the SDB for the example, in a non-graphical style.

2.1 Sessions

The user submits requests to the SM within a session. Other users can share this session and exchange data within it; moreover different sessions can share the same process.

In the example below a new session has been created (service `NewSession()`).

```
==========================================================

Last operation: Created session 1 [29 Feb, 11:21:26]

------------------------------

Session 1 (no queries)

==========================================================
```

The user opens the connection with the WebSM by entering a session. The home page of the WebSM presents a list of available sessions, together with links that can be followed to enter each session.

2.2 Applications, Processes and Agents

An application is an executable program.The command `startAppl(s,a)` begins the execution of the application a using a new process p; it also stores in the Sessions Database (SDB) the information related to p.

The applications available via the SM can be on machines different from the one in which the SM operates.

The interaction between the process and the session manager is realized using *agents* that send the inputs to, and receive the output from, the application process via sockets, nfs or other forms of communication. The agents run on the same machine of the SM and communicate with it using the filesystem, in the following way: the SM writes the input of the query on a file (say fin), then the agent communicates the input to the (possibly remote) application, waits for the result and writes it on another file (say fout).

After the execution of the command `startAppl(1,cocoa)` the SDB looks as follows:

```
==========================================================

Last operation: Created process 1 [29 Feb, 11:26:09]

- Process 1 : (CoCoA) serving query 1.1 (history 1.1)

------------------------------

- Session 1 (1 queries)
```

```
1.1 Input: START CoCoA, Output: NULL, PID: 1, submitted

==========================================================
```

As the application process starts, a new query is automatically prepared to save the output related to the initialization of the application.

The WebSM presents to the user a page on which are listed all the applications available. For each application there is some information about the machine in which it resides (power and load of the machine, geographic location, etc.) and the list of the processes that run that application. Each running process has a link; clicking on it the user adds the process to the list of the active processes of the session. Moreover, the user can start a new application (command `startAppl()`) by activating the corresponding link ["START"]. The new process that runs the application will become, automatically, an active process of the session.

2.3 Queries

A query is a tuple `<i, o, Status, Infos>`.The user creates a query within a session `s` using the command `AddQuery(s,i)`, where `i` is the input of the query. A query can be submitted to a process using the command `SubmitQuery(s,q,p)` where `s` is the session, `q` is the query and `p` is the process.

Each query of a session has a status. Here is the list of all statuses and their meaning: **Input**: the query has an input, but it is not submitted to any process; **Submitted**: the input of the query has been submitted to a process, the identifier of the process and the submission time is contained in the `Infos`; **Running**: the process to which was the query submitted is now processing it; **Done**: the process has completed the computation and `o` contains its output;

```
==========================================================

Last operation: Created query 1.2 [29 Feb, 11:27:34]

- Process 1 : (CoCoA) serving query 1.1 (history 1.1)

------------------------------

- Session 1 (2 queries)

1.1 Input: START CoCoA, Output: NULL, PID: 1, submitted [960229,11:26:09]

1.2 Input: X := 1+1, Output: NULL, PID: NULL, input

==========================================================
```

The query 1.1 (the query 1 of the session 1) has been submitted to the process 1 which is now serving it. The query 1.2 is not yet submitted to any process.

Using the WebSM the creation of a query is done by writing down the input in a *text-area* and pushing the button ["NEW QUERY"']. The user can submit the query to a process by selecting it within the list of the active processes of the session.

```
==========================================================

Last operation: Served query 2.4 by process 3 [29 Feb, 12:01:39]

Process 1 : (CoCoA) ready (history 1.1 1.2 1.4)

Process 2 : (CoCoA) ready (history 2.1 2.2 2.3)
Process 3 : (Singular) ready (history 1.3 1.5 2.4)

-------------------------------

Session 1 (5 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done

1.2 Input: X := 1+1, Output: OK, PID: 1, done

1.3 Input: START Singular, Output: Wilkommen zu Singular, PID: 3, done

1.4 Input: X, Output: 2, PID: 1, done

1.5 Input: X := x+y, Output: OK, PID: 3, done

-------------------------------

Session 2 (4 queries)

2.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 2, done

2.2 Input: X := 1+2, Output: OK, PID: 2, done

2.3 Input: X, Output: 3, PID: 2, done

2.4 Input: X, Output: x+y, PID: 3, done
===============================================================
```

In the query 1.2, the user asked the process 1 to compute the expression 1+1 and to store the result in the variable X.

In the query 1.4 the user asked the process 1 to print the value of the variable X.

Something similar happened for the queries 2.2 and 2.3. The outputs of the two (identical) queries 1.4 and 2.3 are different because the queries have been submitted to two different processes, even if they run the same application.

The interaction with p_3 shows how two users can communicate data by sharing the same process. The first user puts in the variable X of p_3 the polynomial x+y (query 1.5) and the other user looks at it by asking the process p_3 for the value of the variable X (query 2.4).

2.4 Reports

An important service of the SM is the production of reports. Each user can ask the Session Manager for a report on one or more sessions. Different users can ask for different reports on the same SDB and on the same sessions.

The reports may differ from the order of presentation of the queries(by submission time or by creation time), by the portion of the view (query of a process, query of all the processes of an application, query range). Furthermore the user can ask for a report containing just the output of a particular query. Often the outputs are very big and the user would prefer to navigate them in a hypertextual style. For example:

a matrix of polynomials could be reported as a matrix (HTML table) of hyperlinks, each of which can be represented by a string that summarizes the polynomial in that entry (e.g.: it gives the length of the polynomial). The user interested in seeing the entire polynomial can click on the link and receive the document containing the polynomial.

As WebSM runs on a WWW client, the navigation on the data is easy to perform. Obviously the applications should give sufficient information to the WebSM for the production of hypertextual outputs (for example in HTML). It is interesting to note that the applications can cooperate with their agents in producing hypertextual documents. The HTML makes it easy to write documents in "pretty" style. For example the HTML 3 will support "mathematical environment".

2.5 Scheduling of the Queries

The SM communicates each submitted query to the associated process by running the agent of the process. Periodically the SM checks to see if some process has completed a query (i.e.: its agent has written the result on the output file); if so, the SM copies the result of the query in the field `Output` of the query itself, and sets the status of the query to **Done**.

Then, the SM gets the next query queued for that process and contacts the agent for its computation.

2.6 Crash Recovery

Since all the sessions are stored in the machine on which the SM runs, the crash of a client's machine is not a problem. The problem of crash and its recovery is only a problem on the SM machine and the machines which run the processes.

The use of different applications on different machines implies a higher probability of a crash. The SM model provides a simple way to do crash recovery of abnormally terminated processes. The list of all inputs computed by each process (i.e. the history of the process) is stored in the part of SDB devoted to the processes;

when a process abnormally terminates its execution, we can restore it by submitting all the queries processed until its termination.

E.g.: suppose that after the computation of the query 1.2 the process 1 dies.

```
==============================================================

Last operation: Process 1 abormally terminated [29 Feb, 14:17:42]

Process 1 : (CoCoA) defunct (history 1.1 1.2)

-------------------------------

Session 1 (4 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done

1.2 Input: X := 1+1, Output: OK, PID: 1, done

1.3 Input: X, Output: NULL, PID: 1, submitted

==============================================================
```

We can restart the process by submitting the command `RestartProcess(1,1)`.

```
==============================================================

Last operation: Served query 1.3 by process 2 [29 Feb, 14:17:59]

Process 1 : (CoCoA) defunct ... 1.1 1.2
Process 2 : (CoCoA) ready ... 1.4 1.3

------------------------------

Session 1 (5 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done

1.2 Input: X := 1+1, Output: OK, PID: 1, done

1.4 Input: START CoCoA, Output: Welcome to CoCoA, PID: 2, done

1.3 Input: X, Output: Undefined var X, PID: 2, done

==============================================================
```

We note that the query 1.4 has been moved to the new process 2; but that the process 2 has no memory of the assignment done in the query 1.2. We can perform crash recovery by using the command `RecoverProcess` instead of using the command `RestartProcess`.

```
==============================================================

Last operation: Served query 1.3 by process 2 [29 Feb, 14:18:35]

Process 1 : (CoCoA) defunct (history 1.1 1.2)

Process 2 : (CoCoA) ready (history 1.4 1.2 1.3)

------------------------------

Session 1 (5 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done

1.2 Input: X := 1+1, Output: OK, PID: 2, done

1.4 Input: START CoCoA, Output: Welcome to CoCoA, PID: 2, done

1.3 Input: X, Output: 2, PID: 2, done

==============================================================
```

Note that the process has now recomputed the query 1.2 and hence the query 1.3 is correctly answered.

Obviously this approach is not efficient but it is fundamental because it is transparent to the user. Remember that our goal was to produce a common, user-friendly interface to different applications.

Moreover, depending on the application used, the user can speed up possible crash recoveries by "saving the state" of the process and possibly "restoring it"'. Suppose for example that we do some big computation and store its result in a variable, using a command like `X := Hard(A,B,C)`. Also suppose that `V` is the final value of the execution. Then a simple way of helping the SM is to replace the query `X := Hard(A,B,C)` with the new one `X:=V`. In this way if a crash occurs the hard computation will not be redone.

2.7 Educational issues

Since many users can share the same session, our model is useful for teaching purposes. E.g. we can suppose that one user (the teacher) writes down a session and that other users (the students) read it. In this way the WebSM can be used for giving a lesson. Suppose now that the teacher proposes an exercise to the students. The teacher can then write down one query for each student and ask each student to write down the result of ther investigation. Then the session will contain all the exercises written by all the students. We are certain that educators will find many other interesting ways to use the potentiality of a system constructed in this way.

The shared use of a session implies that considerations of ownership of sessions (and of queries inside a session) and of security must be made. For example one might prefer that the only person who can write the lesson is the teacher. Furthermore we want to avoid the possibility that a student could look at the exercise done by another student. A simple solution to this problem is to add the possibility for users to declare themselves, with a password and thus protect their data. This aspect of the SM (and of the WebSM) is currently under development.

REFERENCES

[BI] A.M. Bigatti (1996). Computations of Hilbert-Poincare' Series, To Appear on Journal of Pure and Applied Algebra.

[BU] B.Buchberger (1985).Grobner bases: An algorithmic method in polynomial ideal theory. Recent Trends in Mathematical Systems Theory, (N.K. Bose, Ed.), D.Reidel, Dordrecht, 184-232.

[CNR]A.Capani, G.Niesi, L.Robbiano. (1995) CoCoA, a system for doing Computations in Commutative Algebra,.Available via anonymous ftp from *lancelot.dima.unige.it*

[MA]Bayer D. and M. Stillman (1982-1990). Macaulay: A system for computation in algebraic geometry and commutative algebra. Available via anonymous ftp from *zariski.harvard.edu*

[TC] Pfister, G. The tangent cone algorithm and some applications to local algebraic geometry. In Mora, T. and Traverso, C. (eds.),Effective Methods in Algebraic Geometry, Progress in Mathematics 94, Birkauser Verlag, Basel, 401-409.

[RO] L Robbiano (1988). Introduction to the theory of Grobner bases, Queen's Papers in Pure and Applied Mathematics.

[GMRT] P.Gianni, T.Mora, L.Robbiano, C.Traverso (1994). Hilbert function and Buchberger algorithm preprint.

[Sing]G.M. Greuel, G.Pfister, H.Schonemann, Singular, A Computer Algebra System for Algebraic Geometry and Singularity Theory.

Acknowledgements

Thanks to Gianfranco Niesi and Lorenzo Robbiano for their support during the preparation of the manuscript. Thanks to Tony Geramita for (grammatical) help with the final draft.