Connecting declarative software tools

Declarative tools [for] connecting software

Salvador Lucas
Dep. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
slucas@dsic.upv.es

Summary

- Connecting declarative software tools:
  - The verifying compiler project
  - Concrete problems
  - Interoperability for declarative tools and languages
- Declarative tools for connecting software:
  - Models and logics for Web analysis and development
  - Declarative models for security protocols
- Conclusions and future work

Connecting declarative software tools

Connecting declarative tools

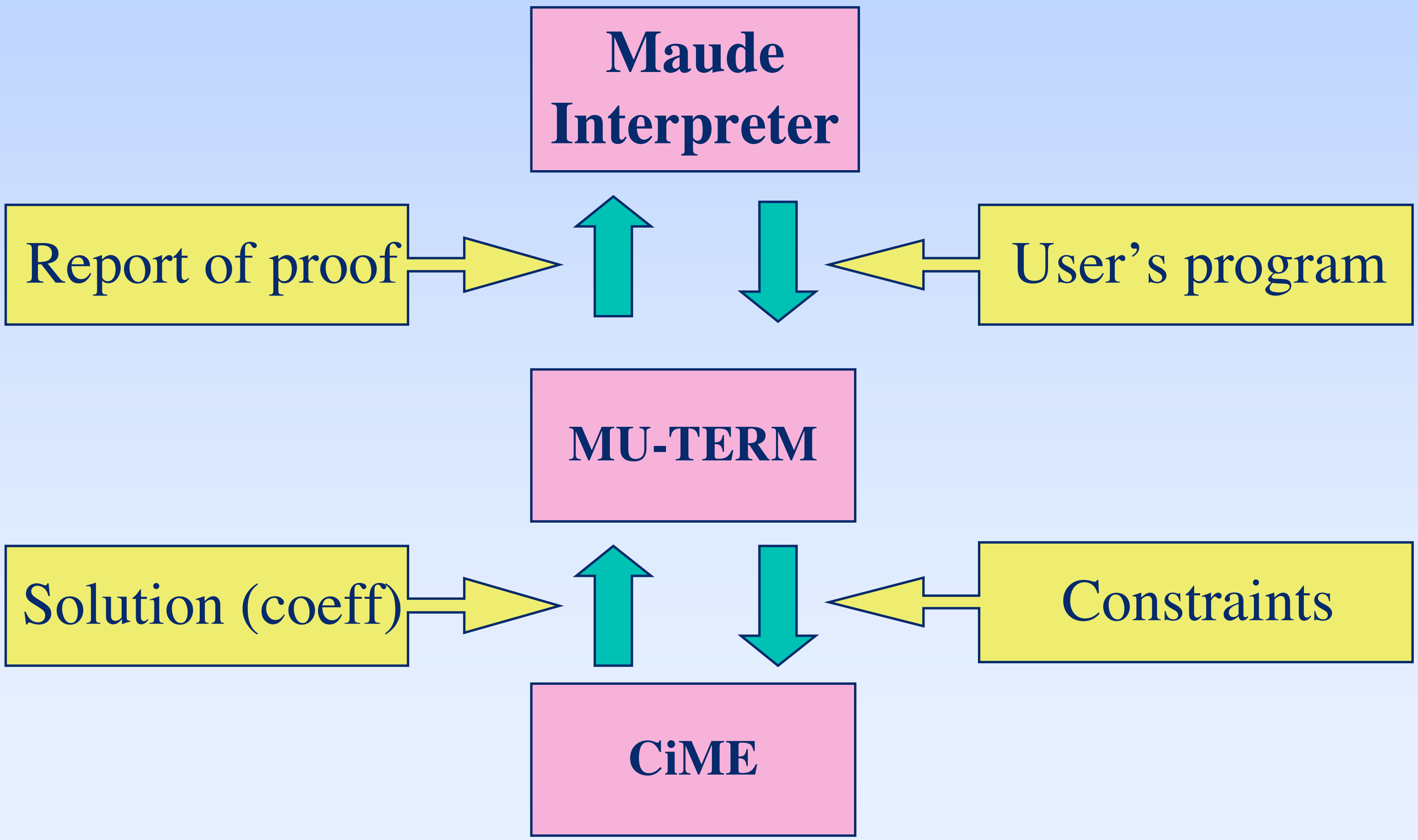
- As part of the 50th anniversary of the Journal of the ACM, an special issue of the journal by highly renowned researchers was published (*Journal of the ACM vol 50, issue 1, January 2003*)

- The aim was to establish the **most important challenges in Informatics and Computer Science for the XXI century**

Connecting declarative tools

- The verifying compiler: a grand (although classic!) challenge revisited by T. Hoare
- Program verification, program debugging, and program analysis will be essential components of such a tool
- Its effective development will require an **incremental** and **cooperative** effort from different work teams **all around the world**

Motivation: declarative languages

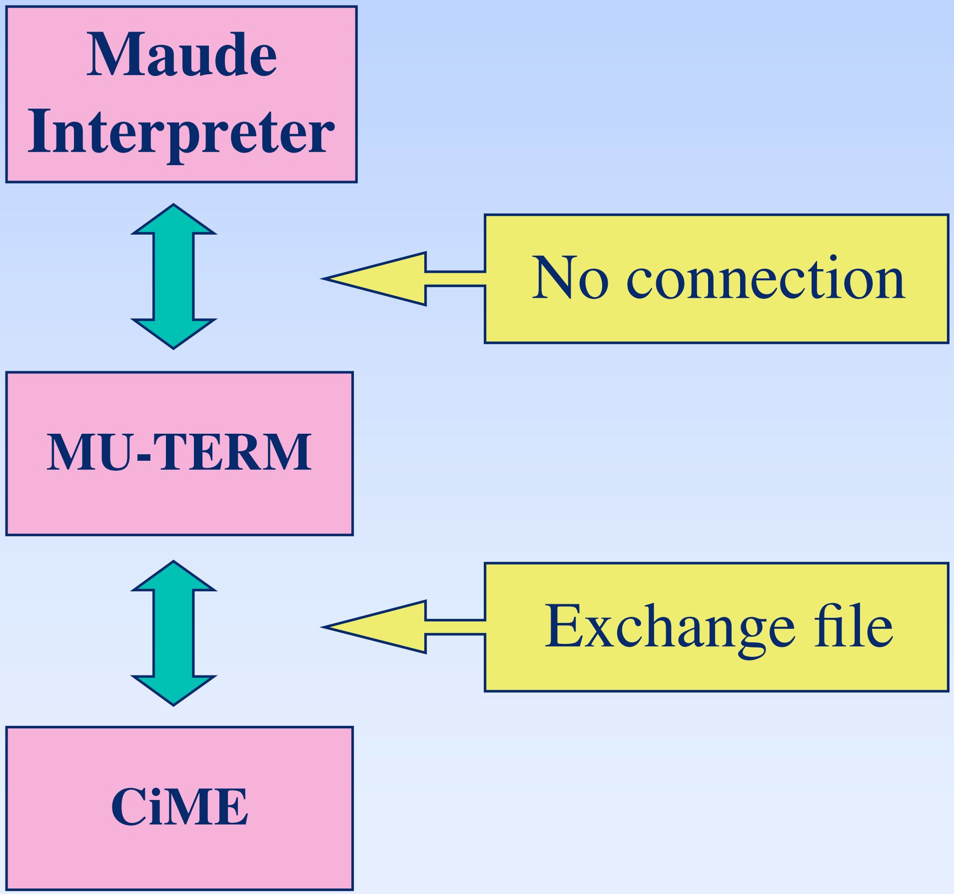


Motivation: declarative languages

*How to **connect** these tools for*
automatically
proving termination of such programs?

Connecting software tools: concrete problems

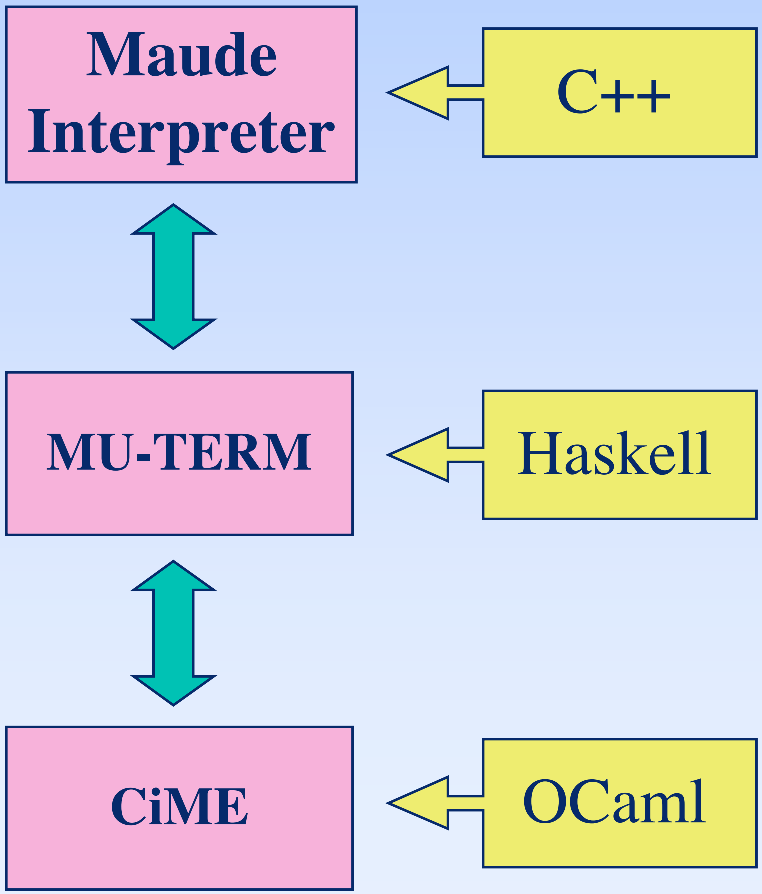
Connecting tools: concrete problems



Connecting tools: concrete problems

Data structures:

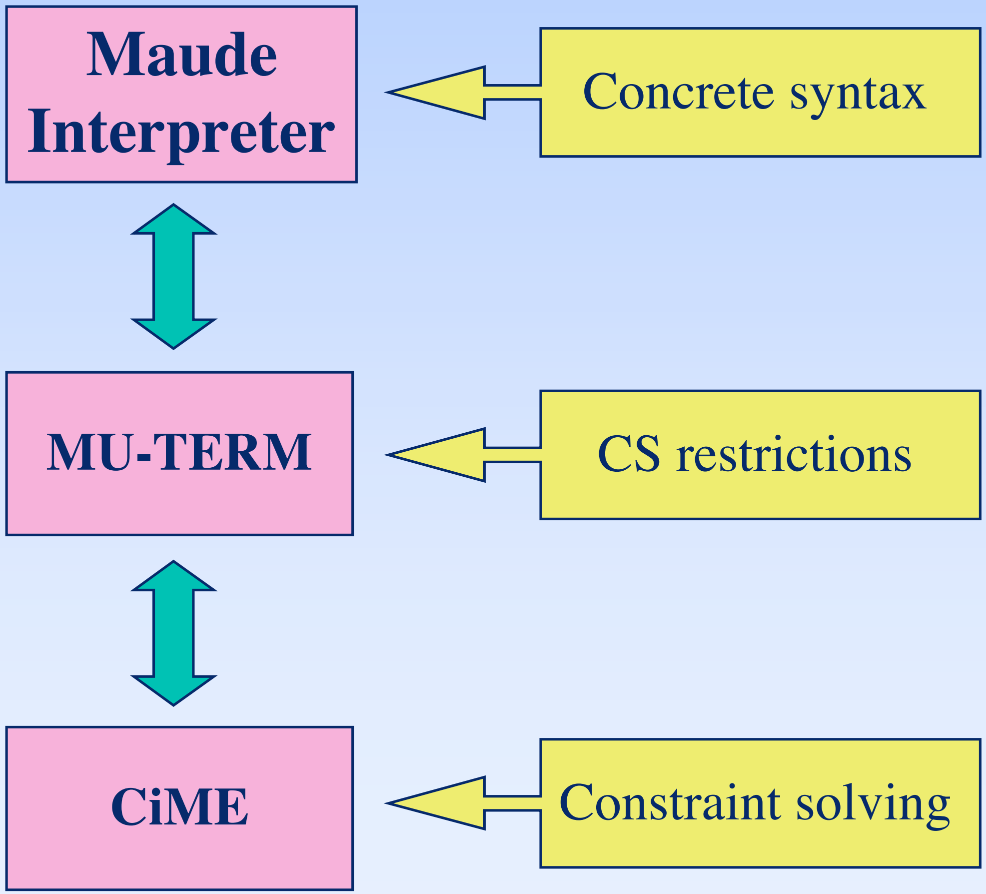
Although they could be linked as object modules, the data representations should be (made) compatible for exchanging data through primary memory



Connecting tools: concrete problems

Distributed:

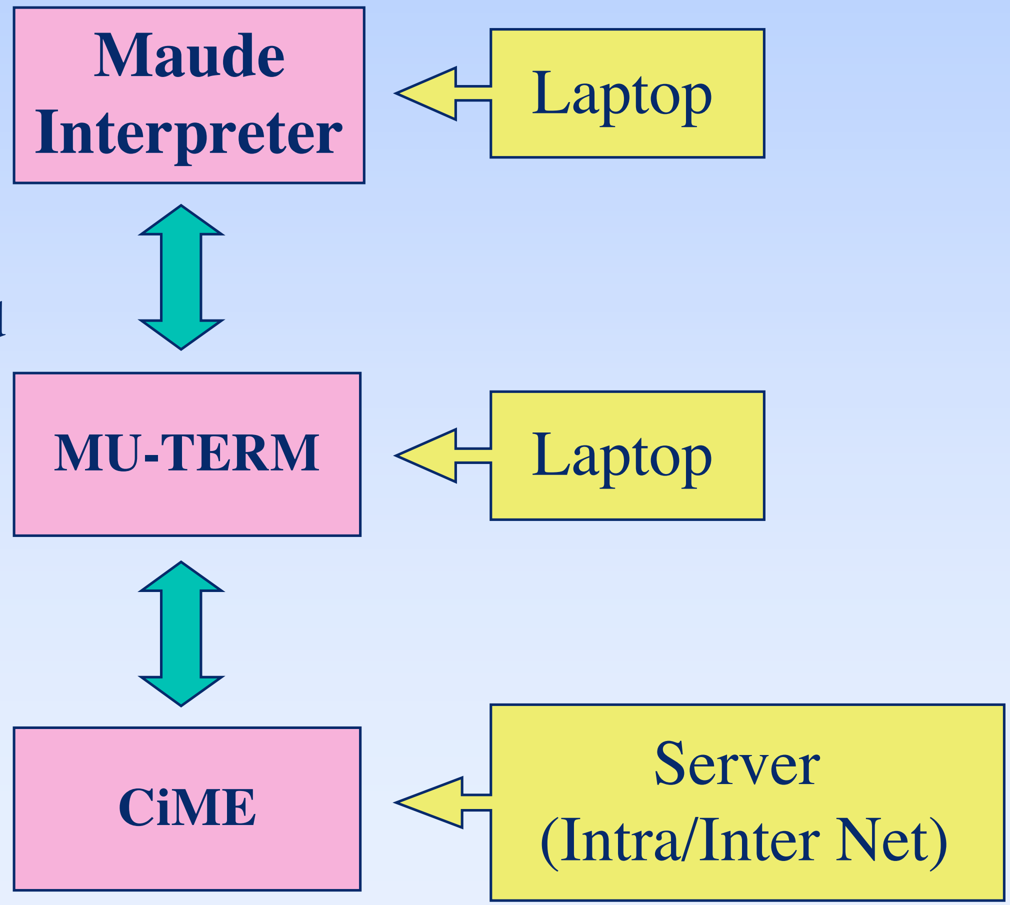
Proofs of termination of Programs involve different kinds of knowledge and expertise. Combining different tools to prove termination is often necessary



Connecting tools: concrete problems

Efficiency:

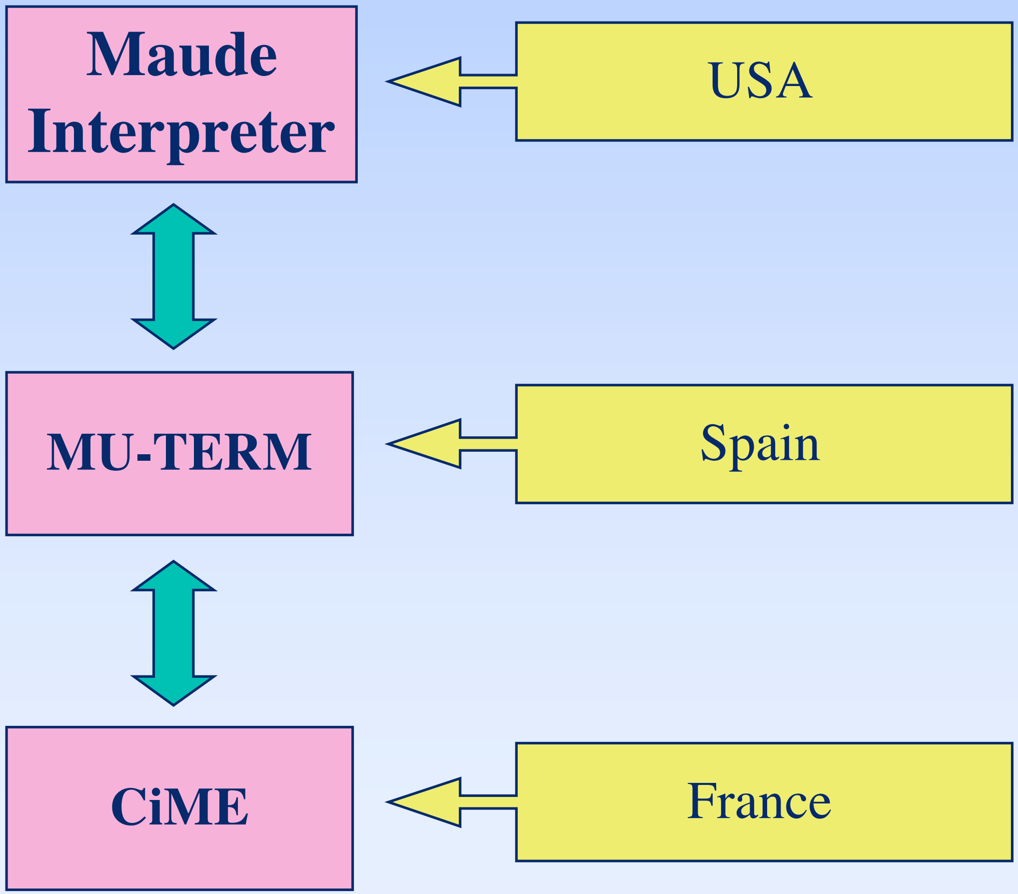
Proofs of termination involve search problems which are costly. Having specialized servers devoted to prove termination can be useful



Connecting tools: concrete problems

International:

Maude is developed and maintained (mainly) by the UIUC and SRI at USA;
MU-TERM has been made at the UPV (Spain)
CiME is being developed at the U. Paris VII (France)



Connecting applications: interoperability

Connecting applications

- Interoperability: making possible for a program on one system to get access to programs and data on another system

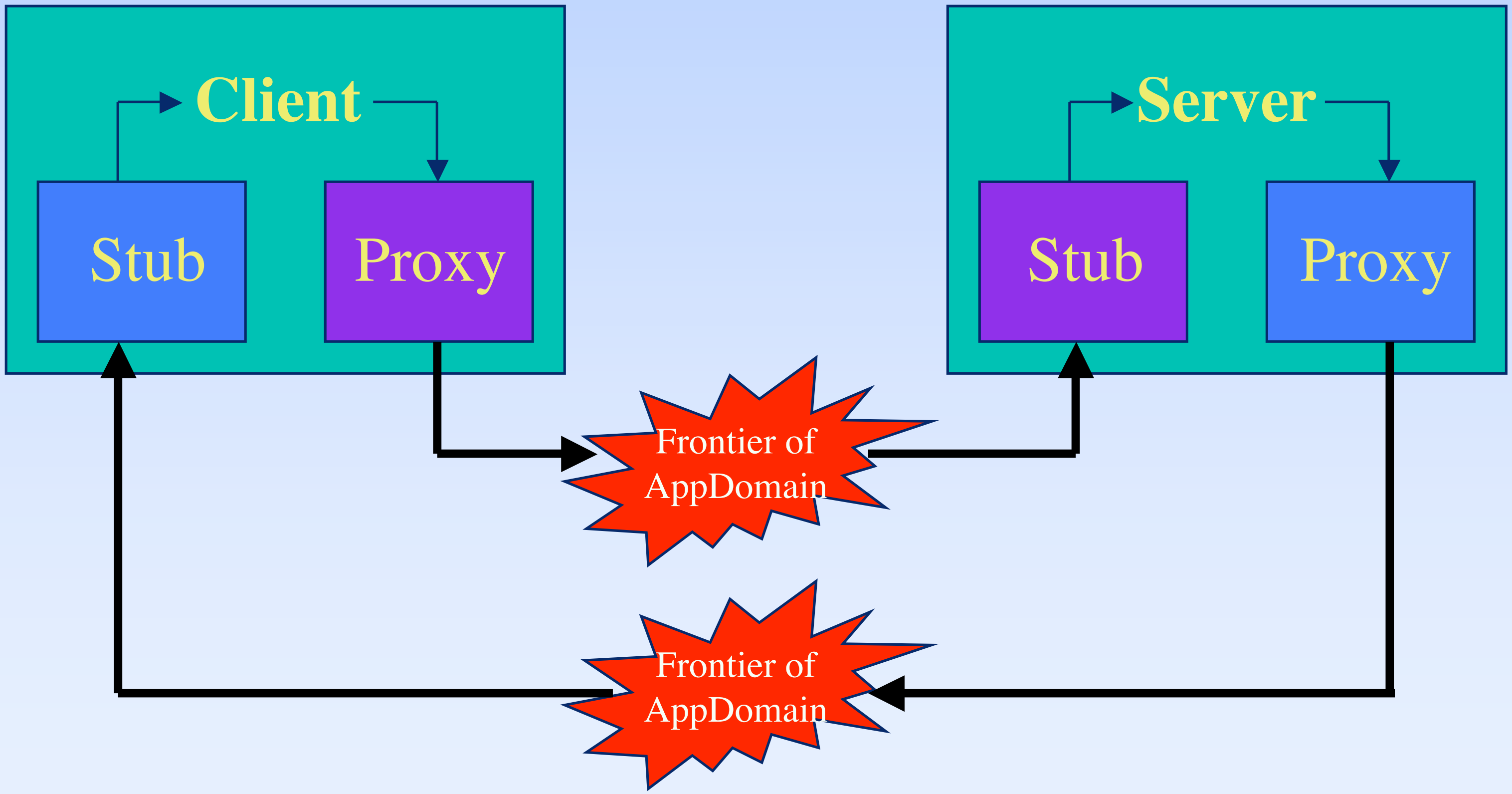
- Solutions: *Middleware systems*, e.g.,
  - COM
  - .NET
  - XML WWW Services

Connecting applications

- Example: .NET:
  - A core language (CLR) provides an abstract machine to implement more sophisticated languages:
    - C++ (or C#),
    - Java (or Java#)
    - ML,
    - Haskell (Mondrian), etc.
  - The implementations can use a number of libraries (for GUIs, **remote access**,…)

Connecting applications

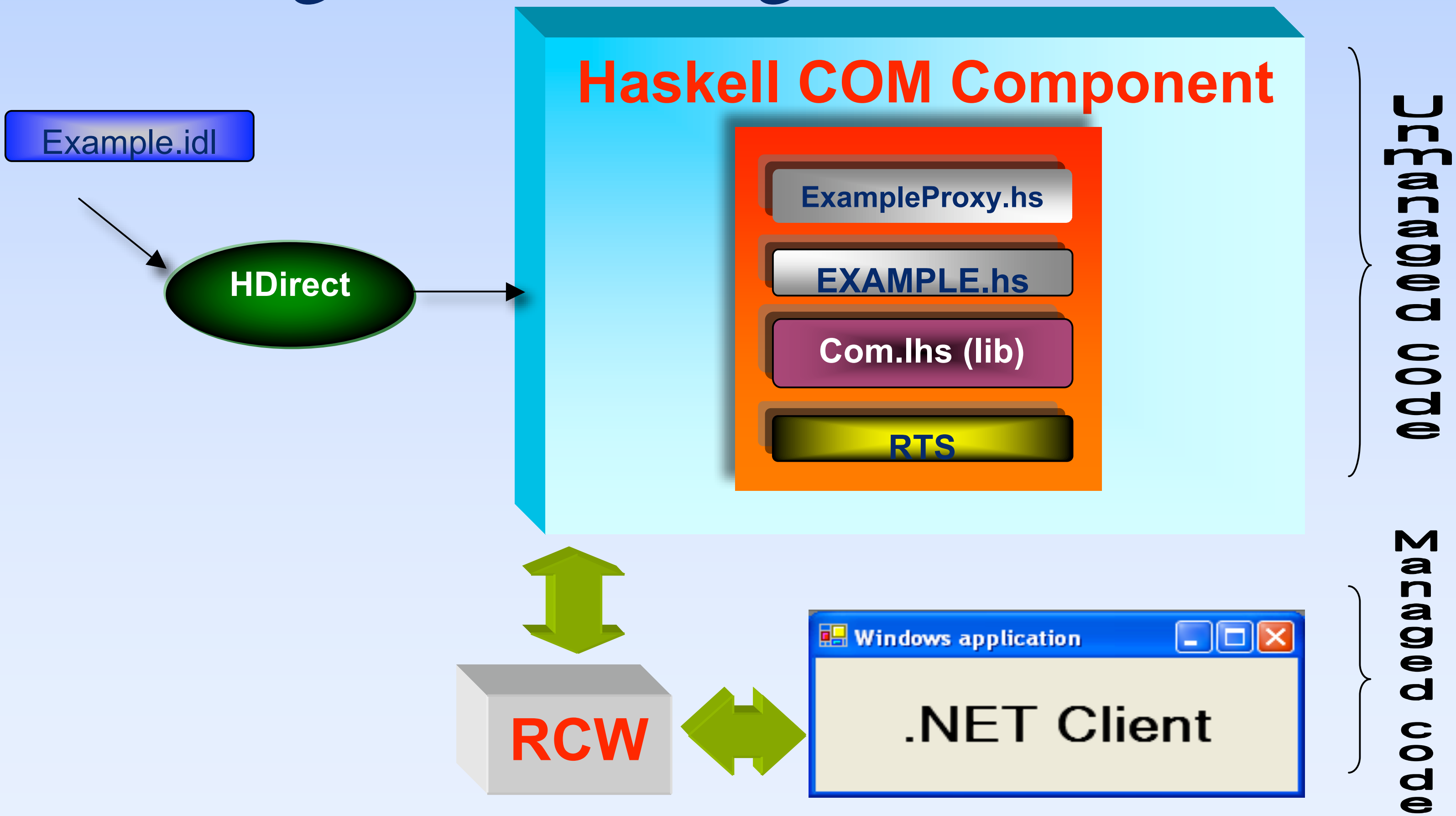
- .NET Remoting:



AppDomains represent **local** or **remote** applications

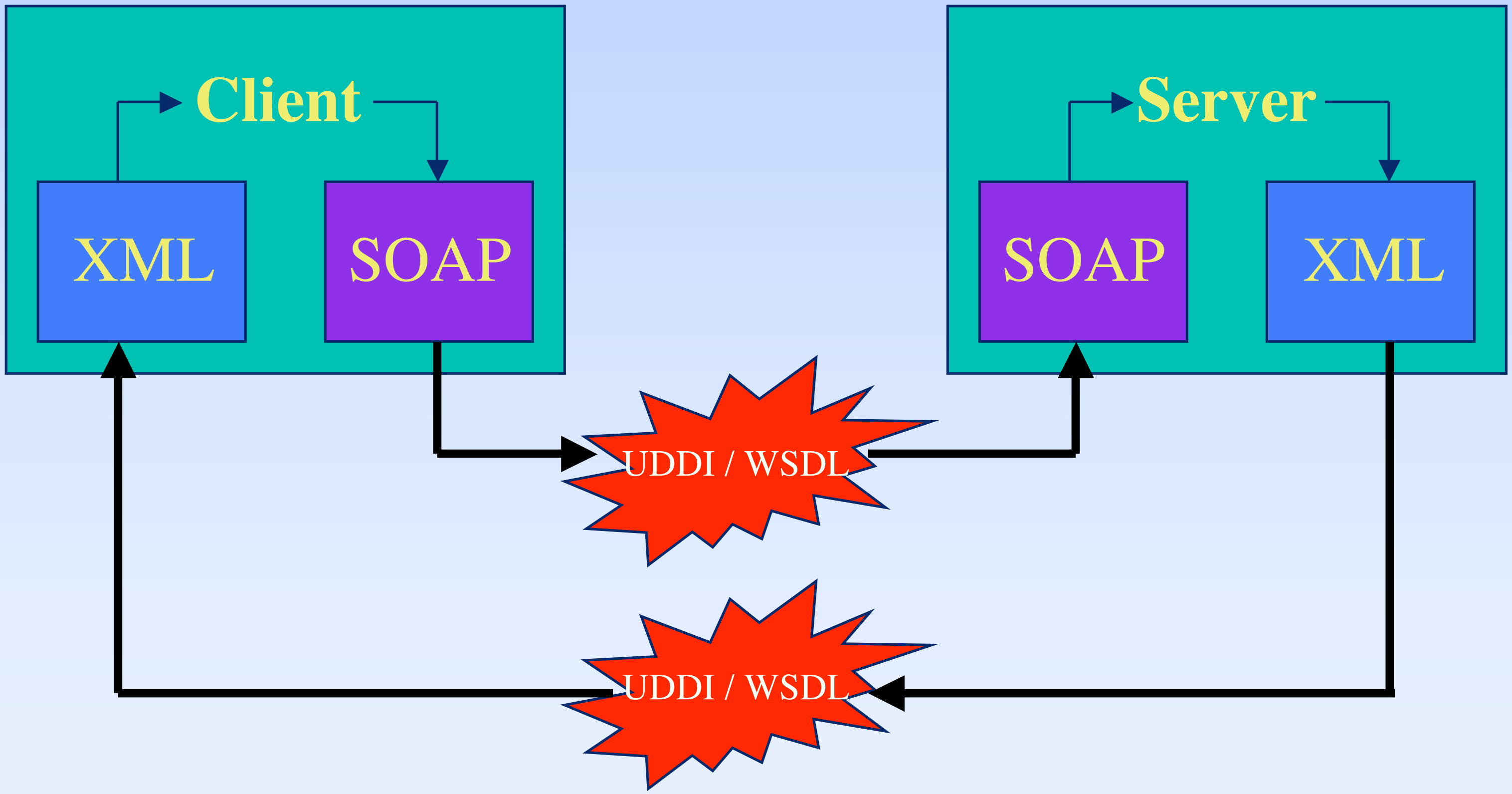
Connecting applications

- Joining .NET through COM:



Connecting applications

- WWW services:



Connecting applications

- Common problems
  - Exchanging data
  - Defining remote services
  - Finding external applications / servers
  - Implementing remote calls
  - Receiving results of remote calls

Connecting software tools: concrete actions

Connecting applications: actions

- TPDB
  - Recent common format for TRSs and termination problems:
    - Conditional equations / rules
    - Strategies
    - Type of problem (TRS, SRS, LP, …)

Connecting applications: actions

- Add information for specifying proofs
  - Simple / C_ε / DP-Simple termination
  - Constraint solving
  - Modular structure
  - Heuristics (and its combinations)
  - Ad-hoc partial / external proofs
- Use of XML for producing input / output information on proofs (e.g., for certification purposes)

Connecting applications: actions

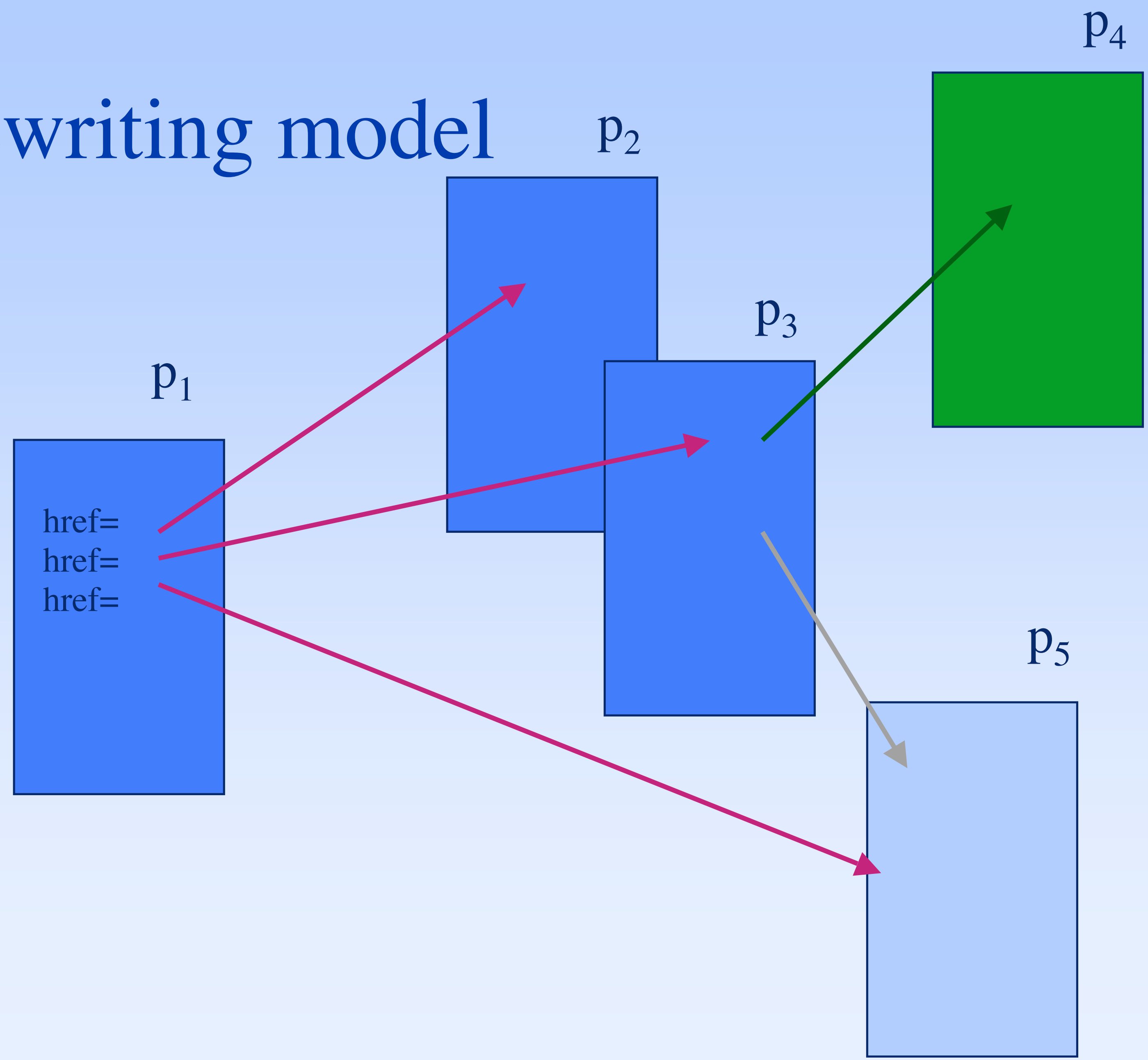
This is an ambitious project which should eventually be agreed / addressed by the interested community.
Coordination with some technical groups (e.g., IFIP WG 1.6 or 1.3,…) would be interesting / desirable

Declarative tools for connecting software

Declarative tools for connectivity

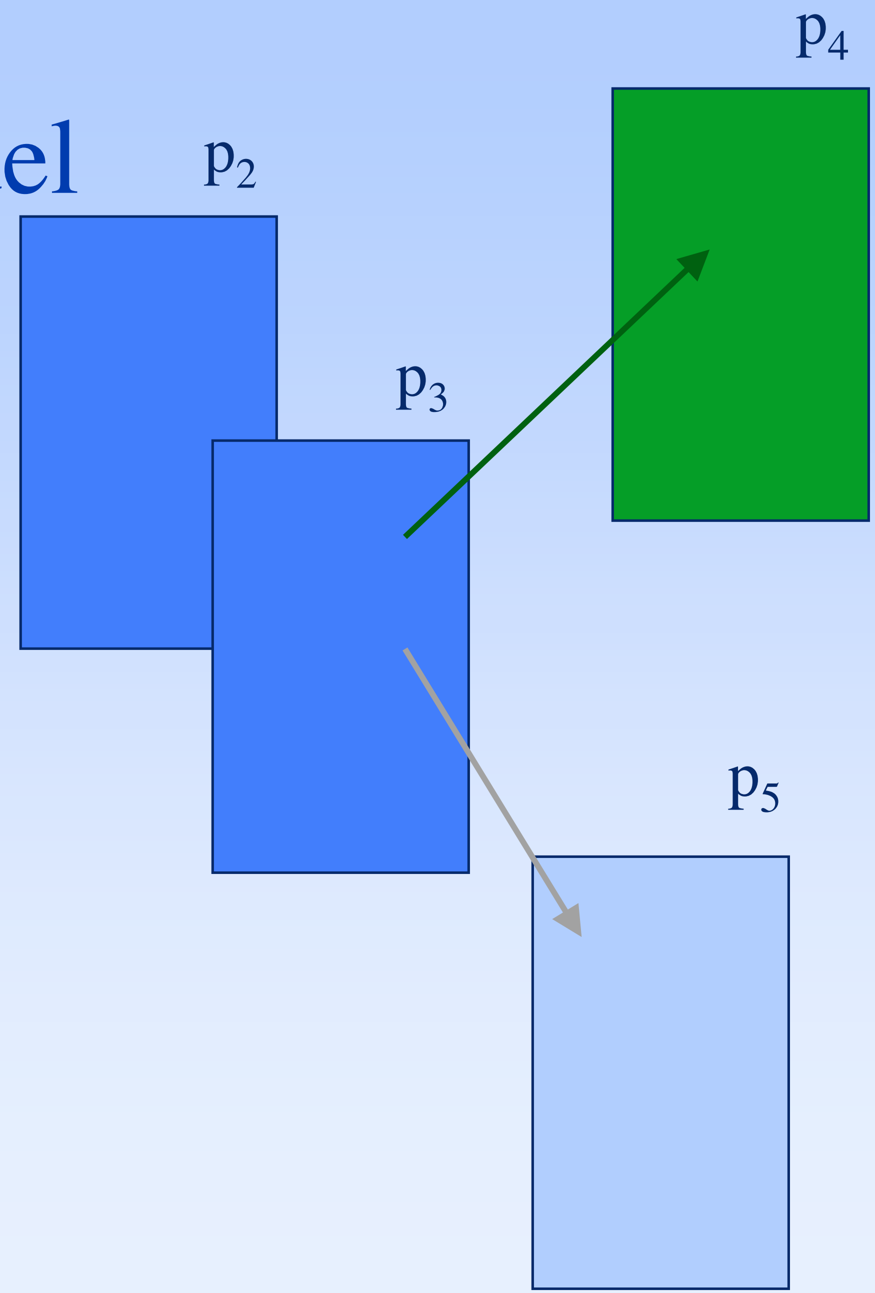
- Web site: a collection of connected Web pages
- Dynamic modeling: focus on the transitions between Web pages

Rewriting model



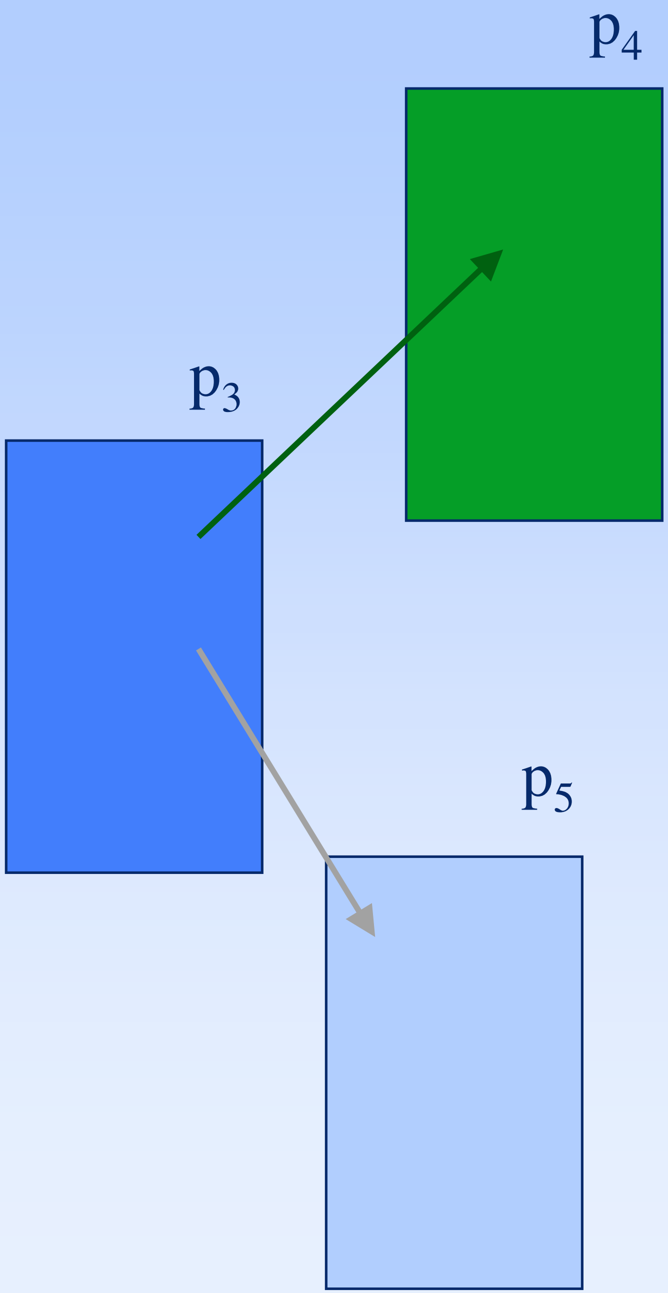
Rewriting model

$p_1(U) \rightarrow p_2(U)$
$p_1(U) \rightarrow p_3(U)$
$p_1(U) \rightarrow p_5(U)$



Rewriting model

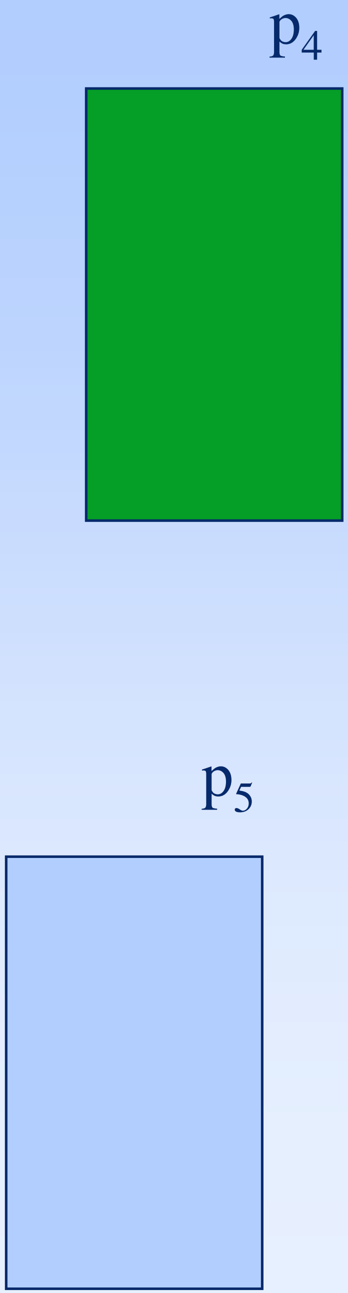
$p_1(U) \rightarrow p_2(U)$
$p_1(U) \rightarrow p_3(U)$
$p_1(U) \rightarrow p_5(U)$



Rewriting model

$p_1(U) \rightarrow p_2(U)$
$p_1(U) \rightarrow p_3(U)$
$p_1(U) \rightarrow p_5(U)$

$p_3(\mathbf{u}) \rightarrow p_4(\mathbf{u})$
$p_3(\mathbf{u'}) \rightarrow p_5(\mathbf{u'})$



Rewriting model

- Term Rewriting System (TRS):

$p_1(U) \rightarrow p_2(U)$
$p_1(U) \rightarrow p_3(U)$
$p_1(U) \rightarrow p_5(U)$

$p_3(\mathbf{u}) \rightarrow p_4(\mathbf{u})$
$p_3(\mathbf{u'}) \rightarrow p_5(\mathbf{u'})$

- Rewriting theories: first order logic (with variables ranging on terms) together with a binary predicate R(x,y) associated to a TRS R:
  - $R(x,y) = x \rightarrow y$: one-step rewriting theory
  - $R(x,y) = x \rightarrow^* y$: rewriting theory

Rewriting model and logics

- **Example:** there is no ‘disconnected’ page:

$$\text{™}y\ \exists x\ ((x \neq y) \wedge ((x \rightarrow y) \vee (y \rightarrow x)))$$

  where ‘=‘ is the predicate R(x,y) associated to the empty TRS

- **Example:** there is no unreachable page (from the ‘main’ page):

$$\text{™}x\ (main \rightarrow^* x)$$

$$\text{™}x\ \exists u\ (main(u) \rightarrow^* x)$$

Rewriting model and logics

- **Example:** "*there is no 'disconnected' page*":

$$™y\ \exists x\ ((x \neq y) \wedge ((x \rightarrow y) \vee (y \rightarrow x)))$$

where '=' is the predicate R(x,y) associated to the empty TRS

- **Example:** "*there is no unreachable page (from the 'main' page)*":

$$™x\ (main \rightarrow^* x)$$

$$™x\ \exists u\ (main(u) \rightarrow^* x)$$

$$™x\ (main(u_1) \rightarrow^* x) \vee \ldots \vee (main(u_n) \rightarrow^* x))$$

Rewriting model: improvements

- **Example:** “*no ‘unsafe’ access is possible*”:

™ p ™ q ™ u ™ v $((p(u) \rightarrow^{*} q(v)) \Rightarrow (u=v))$

- This is a **higher-order** sentence which does not belong to any rewriting theory!

Rewriting model: improvements

- This can be solved by introducing a new binary symbol to put together web pages and users as constant symbols: e.g., browse(p,u)

™*p* ™*q* ™*u* ™*v* ((*browse*(*p*,*u*) $\rightarrow^*$ browse(*q*,*v*)) $\Rightarrow$ (*u*=*v*))

- Problem: no decidability results are available!!

Rewriting model: in practice

- Rewriting-based specification languages like **Maude** are well-suited to express dynamic models of Web sites
- In **Maude** a small query language is available (see the proceedings for some examples)
- Some existential queries are even possible on the basis of traversing the (finite) state space by using a breadth-first search strategy

Rewriting model: network protocols

- The **NRL Protocol Analyzer** (NPA) is a well-known tool for the formal specification and analysis of cryptographic protocols

- For the first time a precise formal specification of its ***grammar-based techniques for invariant generation,*** one of the main features of the NPA inference system, has been given

- This formal specification is given within the well-known framework of the **rewriting logic**

Conclusions / future work

Conclusions

- We are approaching the use of software tools with more complex systems (e.g., interpreters of programming languages)
- The combination of different tools with different expertise domain is required here

Conclusions

- Interoperability issues should be systematically considered when developing termination tools
- Rewriting-based logics are useful to model and analyze network systems and Web sites

Future work

- Which are the appropriate **(fragments of) logics** which are useful to specify (and reason about) the dynamic behavior of Web sites?

- How **types**, **strategies**, **conditions**, etc. can help to get a more expressive model or to improve its power from a logic point of view (e.g., recovering decidability of the theories)

Connecting declarative software tools

Salvador Lucas
Dep. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
`slucas@dsic.upv.es`