Composition and Interoperation of Rules

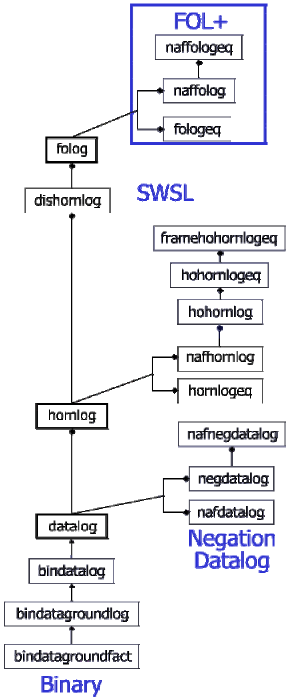
Enrico Pontelli and Tran Cao Son
Department of Computer Science
New Mexico State University

Chitta Baral
Computer Science and Engineering
Arizona State University

Overview

- Motivations
- Proposed Approach
- Formal Syntax and Semantics
  - Pure Language
  - Handling Side Effects
- Implementation
- Conclusions

Motivations



http://www.ruleml.org/modularization/#Model

- Several RuleML languages
  - distinct syntax
  - distinct reasoning mechanisms
- The needs of an agent:
  - Reasoning within a single knowledge base B
    - agent needs to interact with the inference engine relevant for the RuleML flavor of B
  - Reasoning across knowledge bases $B_1,\ldots, B_n$
    - agent's interaction with different inference engines
    - ability to scope reasoning to a specific B_i
    - composition of results from different inference engines

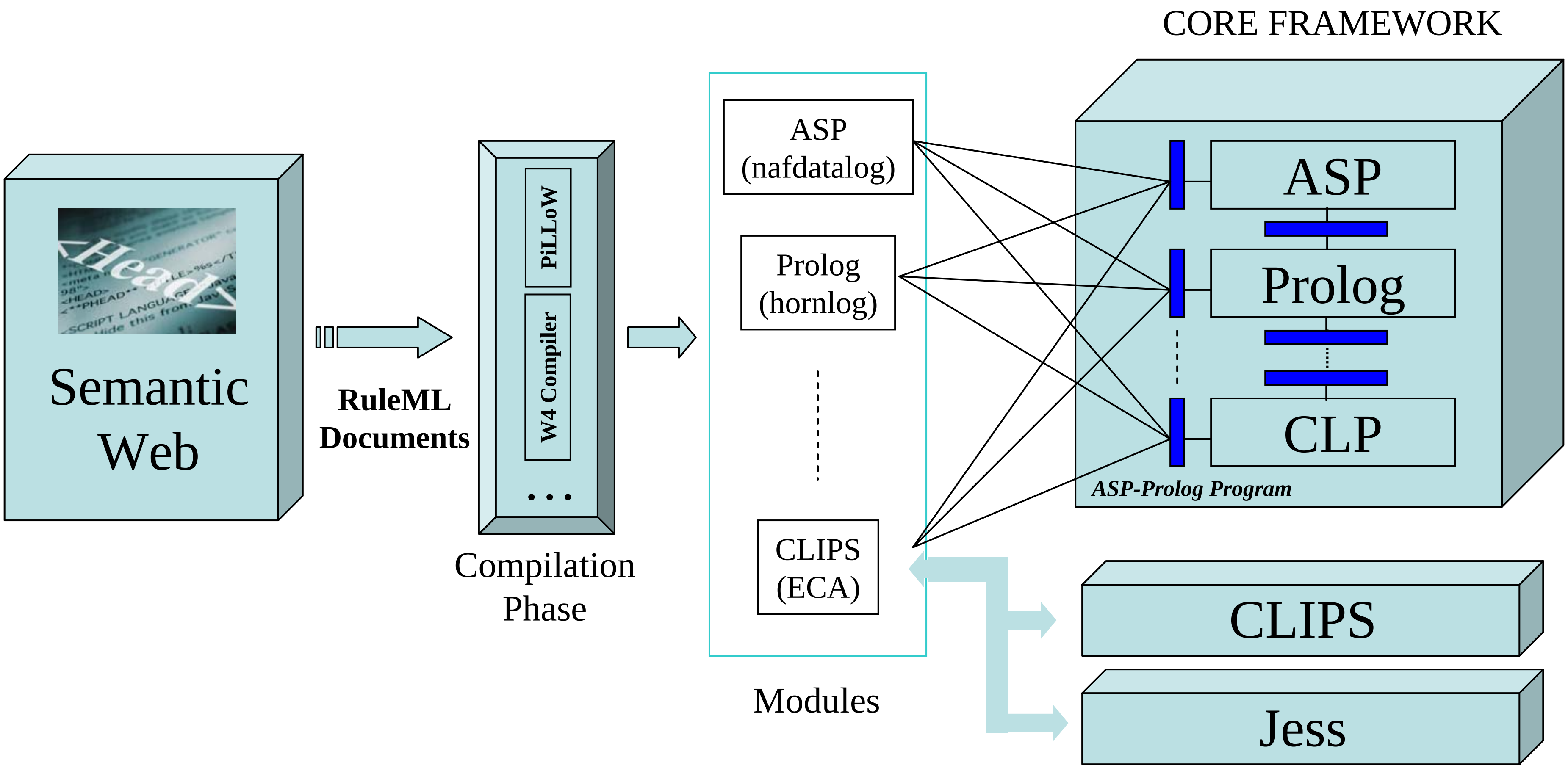
Proposed Approach

- Logic Programming as a common core
  - many RuleML languages can be converted to different flavors of logic programming
- Develop a Logic Programming system that allows integration of modules belonging to different flavors of logic programming
  - standard Prolog
  - Prolog with updates
  - Answer Set Programming
  - Well-founded Model Programming
  - Fuzzy Logic Programming
  - …

Proposed Approach

- Technology
  - ASP-Prolog
    - framework that provides a semantically well-founded integration of Prolog and ASP
  - modules and module hierarchy (import/export lists)
  - PiLLoW (CIAO Library) to access RuleML documents
  - Definite Clause Grammars (DCGs) for conversion of RuleML to logic programming

Architecture



Language Syntax

- Result of RuleML compilation
- $\langle \mathcal{F}, \Pi, \mathcal{V} \rangle$ Signature
  - $\Pi = \Pi_u \cup \Pi_d$ *(user-defined and built-ins)*
    - *built-ins: assert, retract, model*
  - Literals
    - $p(t_1,\ldots,t_n)$ *(atom)*
    - not $p(t_1,\ldots,t_n)$ *(naf-atom)*
    - $t{:}\ p(t_1,\ldots,t_n)$ *(qualified atom)*
  - Rules:

$$A \leftarrow B_1, \ldots, B_n$$

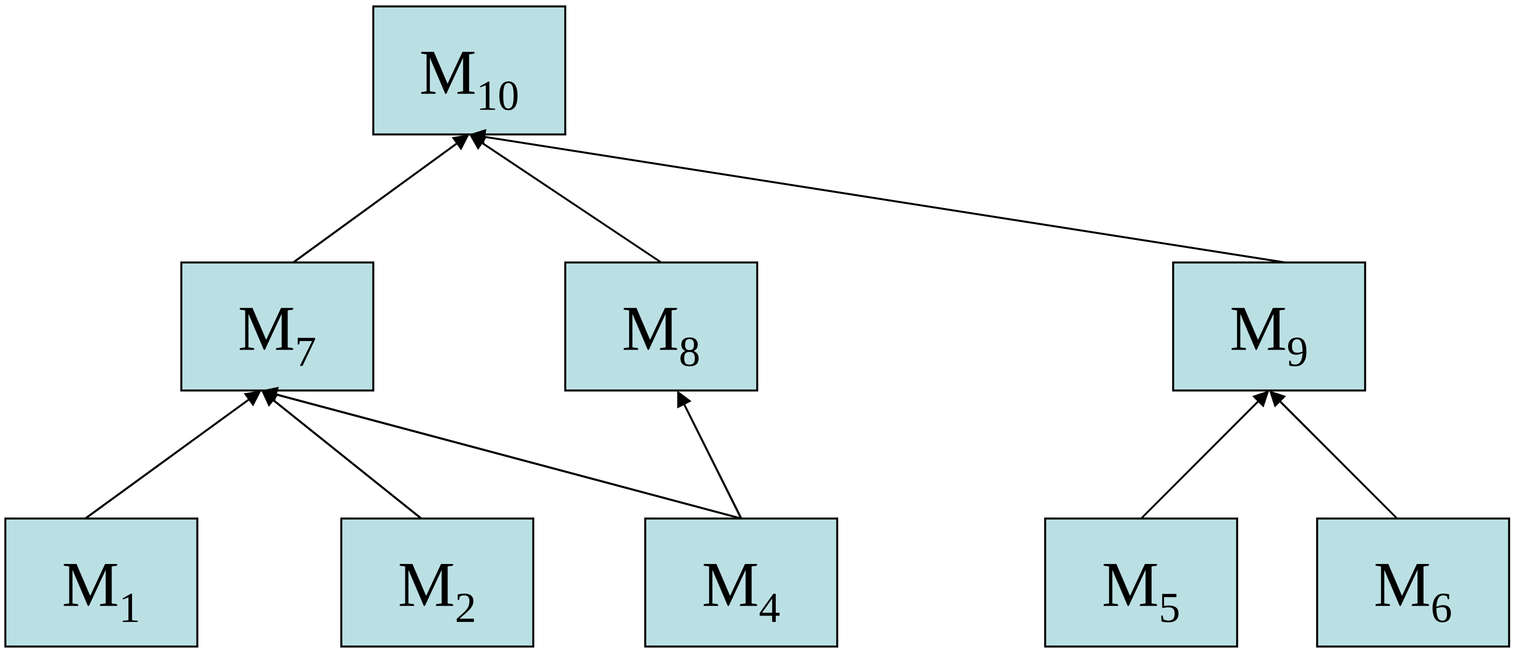
→
- datalog
- ground datalog
- pure Prolog
- impure Prolog
- …

  - Ξ-rules (Ξ = datalog, ground datalog, pure Prolog, …)

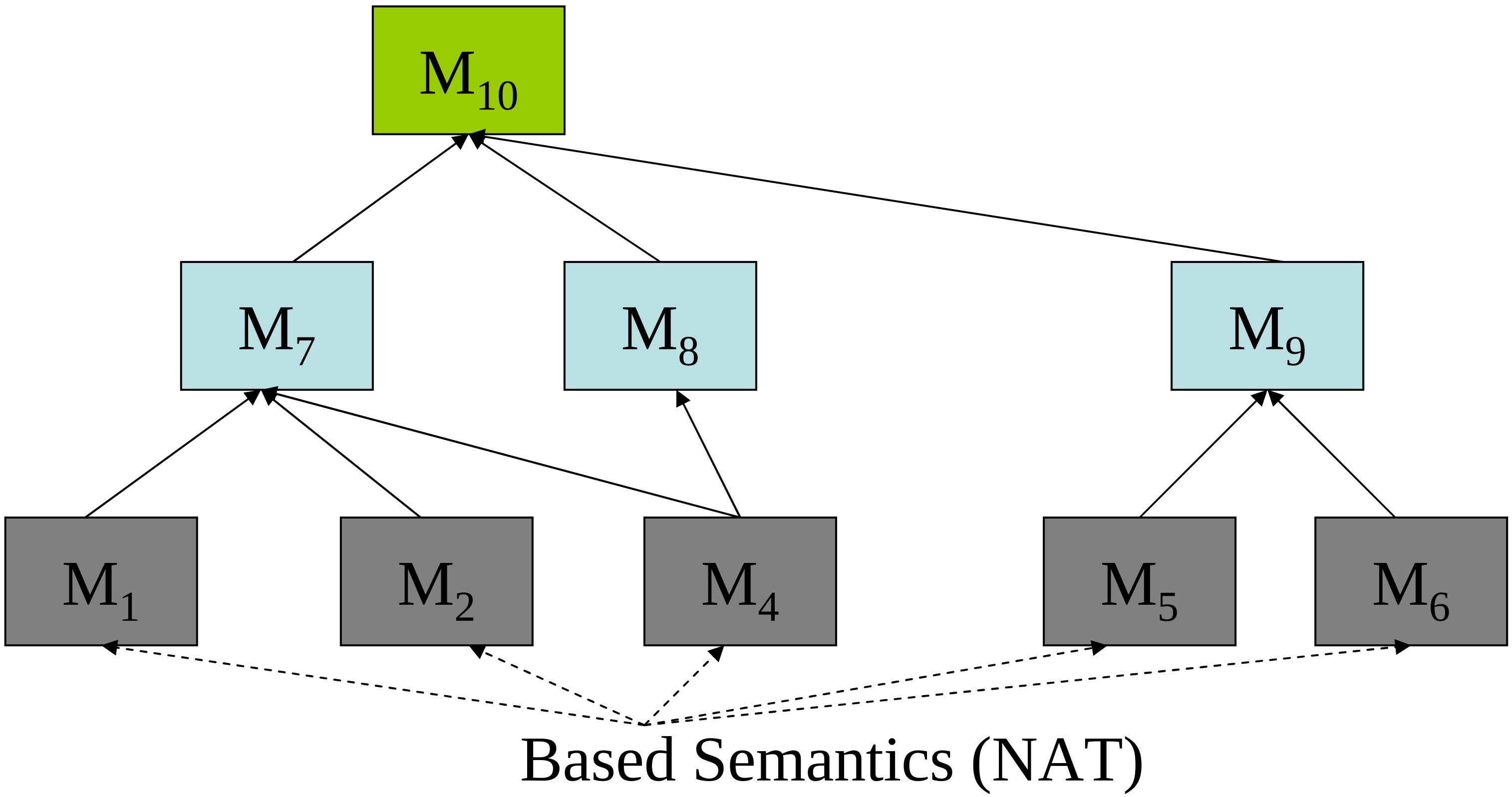
Language Syntax

- Module Structure
  - Module M
    - name(M) *(ground term)*
    - import(M) *(list of ground terms – names of other modules)*
    - export(M) *(list of predicates)*
    - rules(M) *(set of Ξ-rules)*
  - Program P = $\{M_{t1}, \ldots, M_{tn}\}$
    - $name(M_{ti}) = t_i$
  - graph(P) = ({t1,…,tn}, E)
    - $(x,y) \in E$ iff $x \in import(y)$
    - acyclic

Pure Language: Semantics



Pure Language: Semantics



Pure Language: Semantics

- $\tau : H_P \rightarrow 2^{BP}$ (model naming)
- $t_1, \ldots, t_n$ topological sort of graph(P)
- $NAT(T) \subseteq 2^{BP}$ "natural" semantics for a program T
  - T does not contain qualified atoms
  - e.g.,
    - T is a datalog program, NAT(T) is the least Herbrand model of T
    - T is a naf-datalog program, NAT(T) is the set of answer sets of T
- $\mathcal{M}^{\tau}{}_{P}(M_{ti}) \subseteq 2^{BP}$ semantics of module M_{ti}
  - $MR(M, A_1, \ldots, A_k)$ model reduct of module M w.r.t. $A_1, \ldots, A_k$
    - replace each t_i:model(t) with true (false) if $\tau(t) \in A_i$
    - replace each t_i:p with true (false) if $p \in S$ for each $S \in A_i$ (otherwise)
    - replace each t:p with true (false) if $p \in \tau(t)$ ($p \notin \tau(t)$)

$$\mathcal{M}^{\tau}{}_{P}(M_{ti}) = NAT(MR(M_{ti}, \mathcal{M}^{\tau}{}_{P}(M_{t1}), \ldots, \mathcal{M}^{\tau}{}_{P}(M_{ti-1})))$$

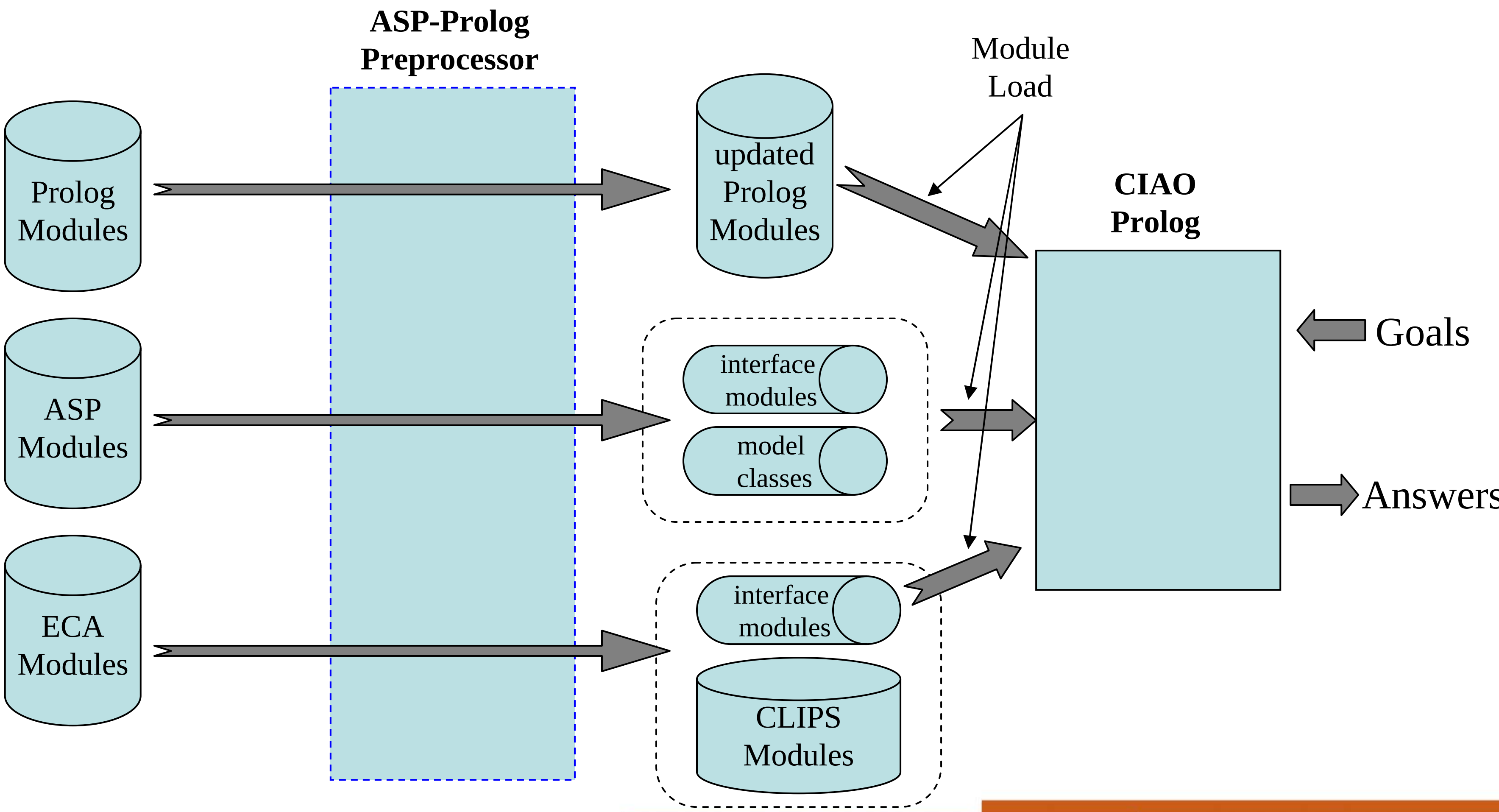
Impure Language

- Presence of assert/retract towards other modules
  - for simplicity, performed in module t_n, impure Prolog module
- Operational Semantics
  - State: (G, θ, P)
  - Transition: $(G, \theta, P) \Rightarrow (G', \theta', P')$
    - select atom A from G, $H \leftarrow Body$ in M_{tn}
      - $G' = (G \setminus \{A\} \cup Body)^{mgu(A,H)}$
      - $\theta' = \theta \circ mgu(A,H)$
      - $P' = P$
    - select t_i:A from G and $H \in S$ for all $S \in \mathcal{M}^{\tau}_P(M_{ti})$
      - $G' = G \setminus \{A\}$
      - $\theta' = \theta \circ mgu(A,H)$
      - $P' = P$
    - select t:A from G and $H \in \tau(t)$
      - $G' = G \setminus \{A\}$
      - $\theta' = \theta \circ mgu(A,H)$
      - $P' = P$
    - select t_i:model(t) from G and $\tau(t) \in \mathcal{M}^{\tau}_P(M_{ti})$
      - $G' = G \setminus \{t_i:model(t)\}$
      - $\theta' = \theta$
      - $P' = P$
    - select t_i:assert(r)/t_i:retract(r) from G
      - $G' = G \setminus \{ t_i:assert(r)/t_i:retract(r)\}$
      - $\theta' = \theta$
      - $P' = P \setminus \{M_{ti}\} \cup \{M_{ti} \cup \{r\}\}$ $[P' = P \setminus \{M_{ti}\} \cup \{M_{ti} \setminus \{r\}\}]$

Handling Non-Logical Modules

- ECA rules
  - simplified view: the model $\mathcal{M}^{\tau}_{P}(M_{ti})$ of an ECA module is the content of the working memory at a stable state
  - ASP/Prolog facts imported as CLIPS facts in working memory
  - Content of the working memory publicized as logic facts

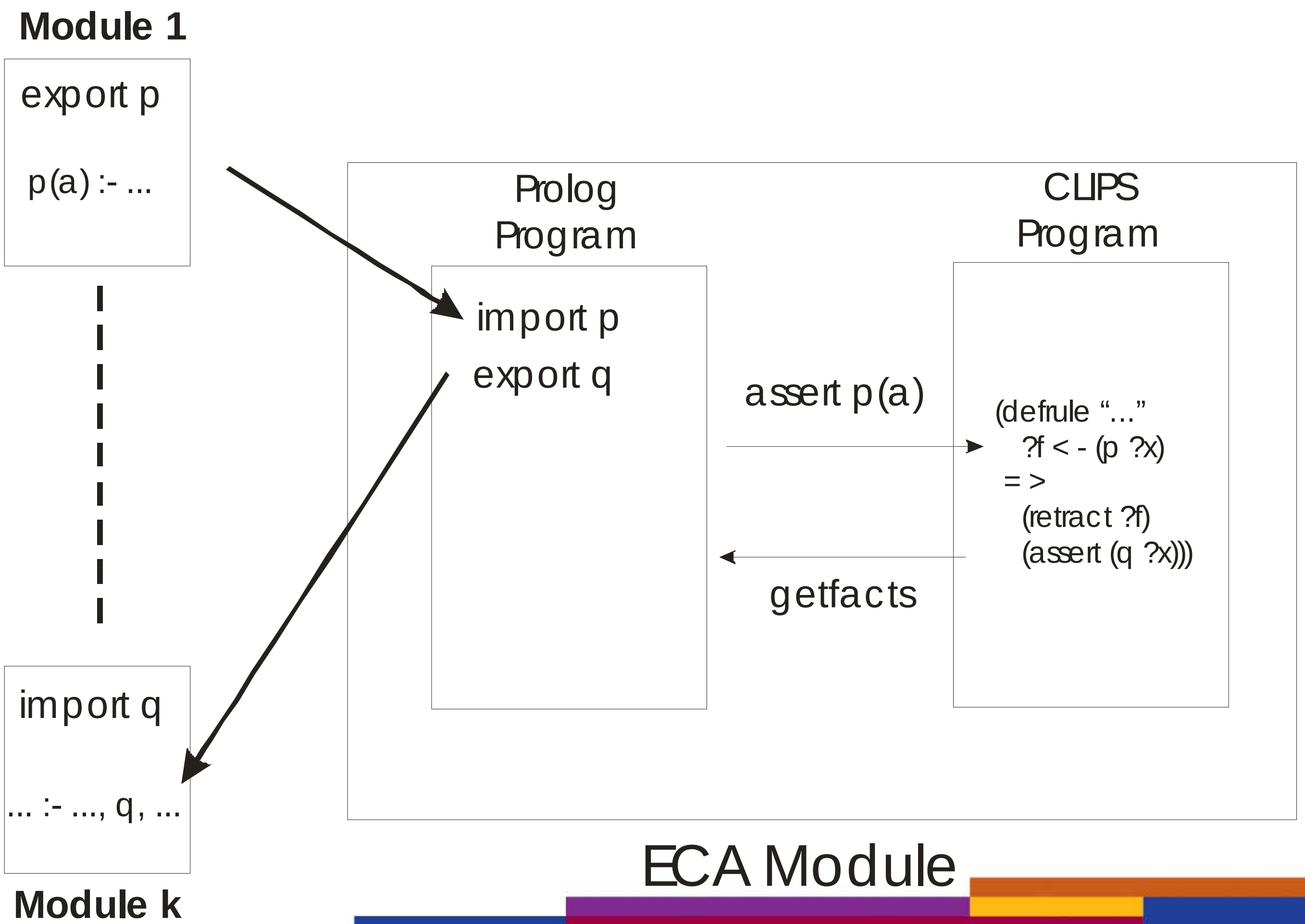
Implementation



Implementation

- Main module (module t_n) CIAO Prolog (top-level)
- Prolog Modules
  - updated to access the newly created CIAO Interfaces of other modules
- ASP Modules
  - compiled to
    - interface module: exports predicates that are defined in the module and built-ins (e.g., assert, retract)
    - model class: a CIAO Class whose objects represent individual answer sets (i.e., sets of ground facts)
    - interface invokes Smodels each time the module is modified; Smodels output converted to instances of the model class
- ECA Modules
  - compiled to CLIPS modules
  - Prolog interface:
    - translates working memory content to Prolog facts
    - implements assert/retract (addition/removal of elements in the working memory of CLIPS)
    - based on CIAO Java interface

Implementation: ECA Rules



Implementation

- Some additional considerations
  - Prolog to be used to
    - access RuleML documents (via PiLLoW HTTP interface)
    - use PiLLoW to convert XML to terms
    - Definite Clause Grammars to parse terms and translate to Prolog/ASP/ECA modules

Conclusion and Future Work

- CIAO Prolog (+ASP, +CLIPS) as a core framework for integration of distinct RuleML flavors
- Logic Programming as a reasoning engine
- Flexibility of Prolog
  - allows to handle conversion to/from RuleML within Prolog
  - allows implementation of sophisticated reasoning mechanisms, e.g.,
    - preferences
    - qualitative reasoning

Conclusion and Future Work

- Complete implementation
  - currently the Prolog+ASP core is completed
  - URL: http://www.cs.nmsu.edu/~okhatib/asp_prolog.html
- Extend the scope of interoperation
  - Extend module structure capabilities
    - inheritance
    - macros
  - RIFRAF