

# Concern Manipulation Environment

IBM T.J. Watson Research Center

IBM Hursley Park, UK

<http://www.research.ibm.com/cme>

1. Promoting Adoption of AOSD
2. Reusable Basis for Tools and Research
3. CME Goals
4. CME Structure
5. Initial CME Tools
6. Initial CME Components
  - a. CCC
  - b. Conman
7. Initial CME Frameworks
  - a. CAT
  - b. CIT
  - c. Puma
8. Initial CME Engines

# Concern Manipulation Environment

## Promoting Adoption of AOSD

To be widely adopted:

### AOSD Requires Extensive Lifecycle Tooling

- Architectural tools
- Design Tools
- Analysis Tools
- Re-engineering Tools
- Programming Language Support
- Analysis Tools

### AOSD Requires Extensive Task-Directed Tooling

- New Development
- Development of Extensions
- Enforcement of Design Rules
- Performance Analysis

### •AOSD Requires Problem-directed Paradigms

- Concern Composition
- Aspect Attachment
- Behavioral Filtering
- Adaptive Routing

# Concern Manipulation Environment

## Reusable Basis for Tools & Research

AOSD Tools gain from broad applicability

Adaptation to a variety of target artifacts

- Requirements analysis
- Architecture
- Design
- Code

Adaptation to a variety of artifact representations

- UML/XMI
- Compiled Java
- Java Source
- C#

Adaptation to a variety of operating environments

- Native-Java
- Java with Eclipse
- Java/Eclipse with Stellation

# Concern Manipulation Environment Goals

Wide AOSD Deployment requires a framework of support to meet all these demands economically

Concern Manipulation Environment (CME)  
is intended to provide such a base:

- Written in Java
- Submitted for approval as an Eclipse open source project\*
- Supporting an Eclipse AOSD environment
- But with independent Java components

\* Disclaimer: All information being released represents IBM's current intent, is subject to change or withdrawal, and represents only goals and objectives.

# CME Structure

## Four-Layer Support Model

### Tools

extensive set of tools

tailored to:

task, lifecycle, paradigm, environment

sparsely covered in initial work

relying on components

### Components

manageable set of components

independent of:

task, lifecycle, paradigm, environment

broadly but sketchily covered initially

adaptable, tailorable behavior

relying on frameworks

*gradual migration*

narrow support base

supporting:

### Frameworks

language/paradigm-independent access  
and manipulation

ready now

using plug-in engines

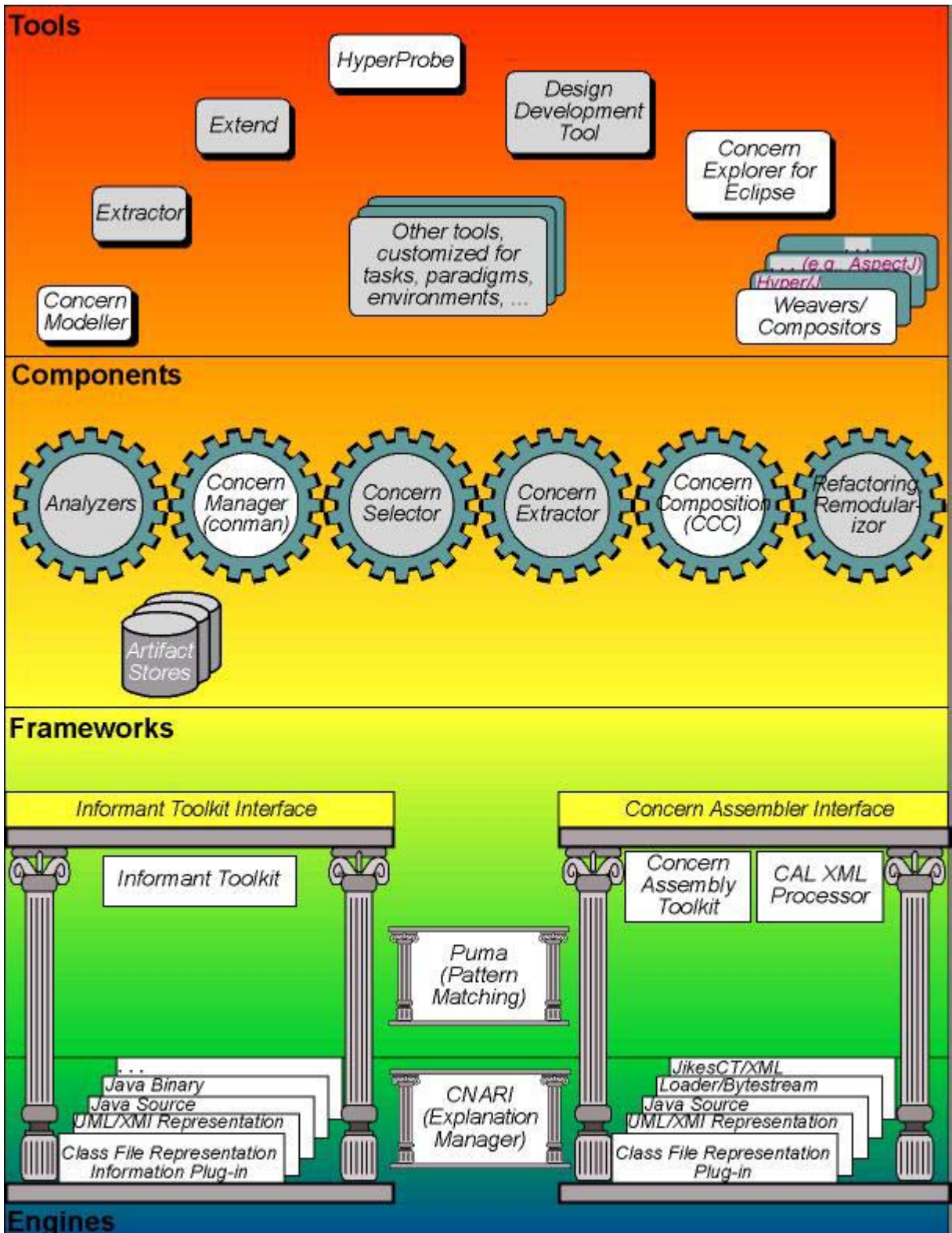
### Engines

Java binary, Java source (Eclipse), XMI

specific to language & representation

subset available

# CME Structure



# Initial CME Tools

## Concern Explorer for Eclipse

Eclipse-based tool integrated with the Eclipse Java Development Toolkit

Provides "Concern Explorer" view

- Definition of concerns and concern groups
  - browsing
  - queries with immediate feedback
- Definition of compositions
  - Dialogues guide user through details
- Building compositions with immediate feedback
- Generation of Ant targets for offline builds
- Navigation and source code access based on concerns and compositions

## Hyper/J2

Stand-alone Hyper/J-like tool for concern definition and composition.

File-based

Same concern modeling and composition as the Concern Explorer for Eclipse

# Initial CME Components

## CCC

### Concern Composition Component

Composes/weaves different concerns in any of the supported artifact languages supported by the CME.

CCC's composition is extremely flexible and supports a wide variety of AOSD paradigms and composition semantics.

Directs the concern assemblers to construct new classes, using explicit formative directions and information about the affected classes.

Supports component selection and correspondence with powerful selection criteria over sets of tuples of components, rather than simple sets of components.

Supports a wide variety of organizations and attachment models appropriate to different AOSD paradigms and applications.

# Initial CME Components

## CCC

### API/Operational Protocol

Specify correspondences describing how outputs are formed from inputs

Formative Correspondences cause the production of outputs

Cross Correspondences describe constraints but do not cause production

Complete reduction of specifications to CAT API.

### Important Concepts

Type (Class/Interface/Classifier), Method, Field – as usual in OO

Concern – closed set of types

Universe – set of type spaces with a common representation

Correspondence – a coordinated n-tuple of items or coordinated item patterns (concerns, types, methods, fields) to be used to form one or more output items

Selection – a specification of how the items in a correspondence are selected/winnowed to form the output, e.g. merge, override, choose

Order – a specification of the ordering relationships among the items being selected, e.g. labels, partial-orders, combination graphs

Structure – a specification of the manner in which each item in a correspondence is attached to the output being formed for the group of inputs, e.g. facet, clone, aspect, associate

Combination Graph – abstract graph for constraining the ordering and flow of control among combined items, e.g. BeforeAfter, summarize

### Extensibility/Flexibility Characteristics

supports wide selection of member-group structures

supports arbitrary partial-ordering of precedence among formative/cross correspondences

supports open-ended set of combination graphs

supports multiple levels of black-box/white-box treatment of inputs

employs plug-in strategy for language-sensitive issues

### Exploitation

CCC is one of the components employed by the Concern Explorer for Eclipse demonstrated at AOSD 2003.

# Initial CME Components

## CCC – Concept Illustration

A formative correspondence from demo (equivalent to UI interactions):

*merge method*

*facet graph(beforeafter): PersonnelPayrollConcern:PersonnelPayroll.\*Manager.removeSubordinate,  
clone label(before): MgrHasPeonsConcern:BusinessRules.ManagerHasPeons.Dictator.ensurePeons,  
as OutSpace:PersonnelPayroll.<1>Manager.removeSubordinate(PersonnelPayroll.Employee)*

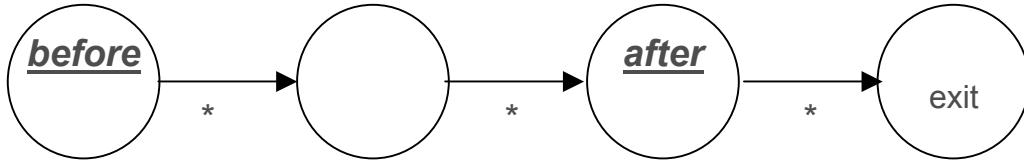
- Merge selection indicates that items in this correspondence are to be used together. Alternatives are override, choose (one)
- Method specifies the kind of items in the correspondence. If “class” were used here the classes containing the methods identified by the patterns are meant.
- Facet structure specifies that this item is to be identical with the one being created. References to it are effectively references to the created item. Clone specifies that the original item has a separate identity but that an equivalent item is intended. Aspect specifies the attachment of an item as a separate object with its own lifetime (with options), etc.
- Graph(beforeafter) indicates that this item plays the unnamed role in a beforeafter graph order <next page>. Label(before) indicates that the item plays the “before” role in its graph. Alternatives also include partial orders.
- Pattern describing items to be included in this set of correspondences. Patterns may include modifiers (keywords), regular expressions, literal material, free variables (\*), bound variables referring to values of free variables (<1>), etc., supported by Puma
- “as” names each item to be produced, using bound variables from the pattern.

# Initial CME Components

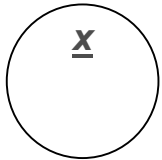
## CCC – Concept Illustration

- Orderings are different ways to specify graphs that can be merged, e.g.

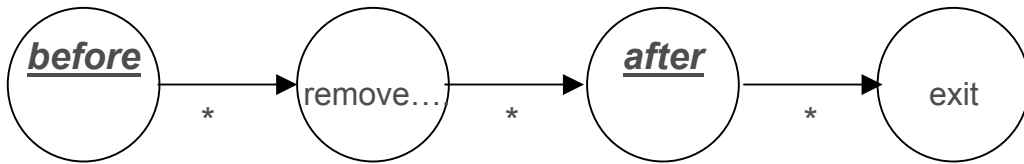
Graph(beforeafter) names this graph:



Label(x) names this graph:



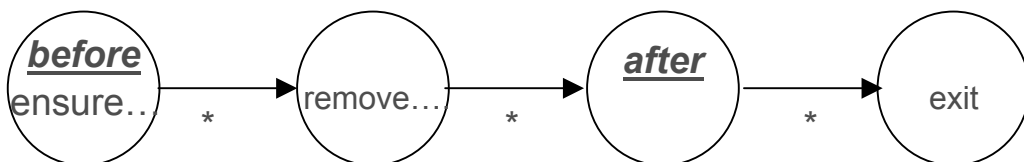
- Each item in a correspondence is placed in its ordering, and their graphs are combined. From preceding page we have:



+



=



- The set of named graphs is easily extended

# Initial CME Components

## ConMan

### Concern Manager

Models software in terms of arbitrary concerns and their interrelationships.

Supports a wide variety of concern structures and software decompositions, such as feature-based, aspect-based, object-based, and rule-based (and supports multiple concern structures simultaneously).

Provides a meta-model of concerns that is intended to be specialized to reflect the semantics of particular AOSD paradigms. This meta-model will be part of the basis for interoperation among different AOSD paradigms.

# Initial CME Components

## ConMan

### Important Concepts

Unit: A (fragment of an) artifact.

Concern: A collection of units and/or concerns that are related semantically.

These collections may be specified *explicitly* or by *pattern*.

Context: A collection of elements, along with any relationships and constraints that apply to those elements.

Composition: A context whose elements are concerns, and whose relationships indicate how the concerns are to be composed.

Relationship: A semantic connection among elements. The elements may be specified *explicitly* or by *pattern*.

Constraint: A semantic restriction that applies to a collection of elements.

Loader: A pluggable component for importing units from an external source.

Builder: A compositor or weaver tool associated with a composition.

### API/Operational Protocol

Load units from external sources.

Create *concern model* by specifying concerns, contexts, relationships, and constraints.

Specify compositions and perform them using associated builders

Import composed results and relationships back into the model

### Extensibility/Flexibility Characteristics

supports arbitrary artifact types, notations, programming languages

meta-model supports open-ended set of concern, relationship, constraint, grouping, and composition types, to support different AOSD paradigms and to promote interoperability

supports *extensional* and *intensional* collection definitions, with multiple evaluation strategies

provides rich, extensible access interface, with built-in connection to Puma

employs plug-in strategy for builders and loaders

### Exploitation

ConMan is one of the components employed by the Concern Explorer for Eclipse demonstrated at AOSD 2003, using a CIT loader and CCC builder.

# Initial CME Frameworks

## CAT – Concern Assembler Toolkit

- Supports the construction of concern assemblers appropriate to a variety of different artifact manipulation engines.
- Provides frameworks for managing those concepts that concern assembly adds to the usual OO concepts that are supported by an engine.

## CIT – Concern Informant Toolkit

- Supports the construction of concern informants appropriate to a variety of different artifact manipulation engines.
- Provides frameworks for managing those concepts that concern information adds to the usual OO concepts that are supported by an engine.

## Puma – Pattern Underlying MAtcher

- Supports selection and correspondence of arbitrary elements with powerful selection criteria over product-sets rather than simple sets of artifacts.
- Provides frameworks and library for handling a wide variety of data organization, retrieval, and query formation strategies, and an open set of query operators and languages.

# Initial CME Frameworks

## CAT

### Important Concepts

Type (Class/Interface/Classifier), Method, Field – as usual in OO

Type Space – closed set of types

Universe – set of type spaces with a common representation

Mapping – description of how references to types, methods, and fields should be transformed when material is copied from inputs to output

Method Combination Graph – Petri-net-like graph for constraining the flow of control among combined methods

Methodoid – characterization of a pattern of content within a method that should be treated as though it had been written like a method for purposes of describing how it is combined.

### API/Operational Protocol

Specify input types to be composed and expected for output from library

Specify methodoids to be identified and made available as methods

Specify types to be created

Specify fields and methods to be copied/renamed into output

Specify outputs defined as method combination graphs

Specify extends/implements relationships

Apply specifications using manipulation engine(s)

### Extensibility/Flexibility Characteristics

API – defined entirely as interfaces. Permits concern assemblers built as facades on top of manipulation engines

Independent sub-frameworks for universe, type space, methodoid, method combination graph, type vector, modifiers

Sub-framework classes are to be subclassed for varied purposes, or can be used as are.

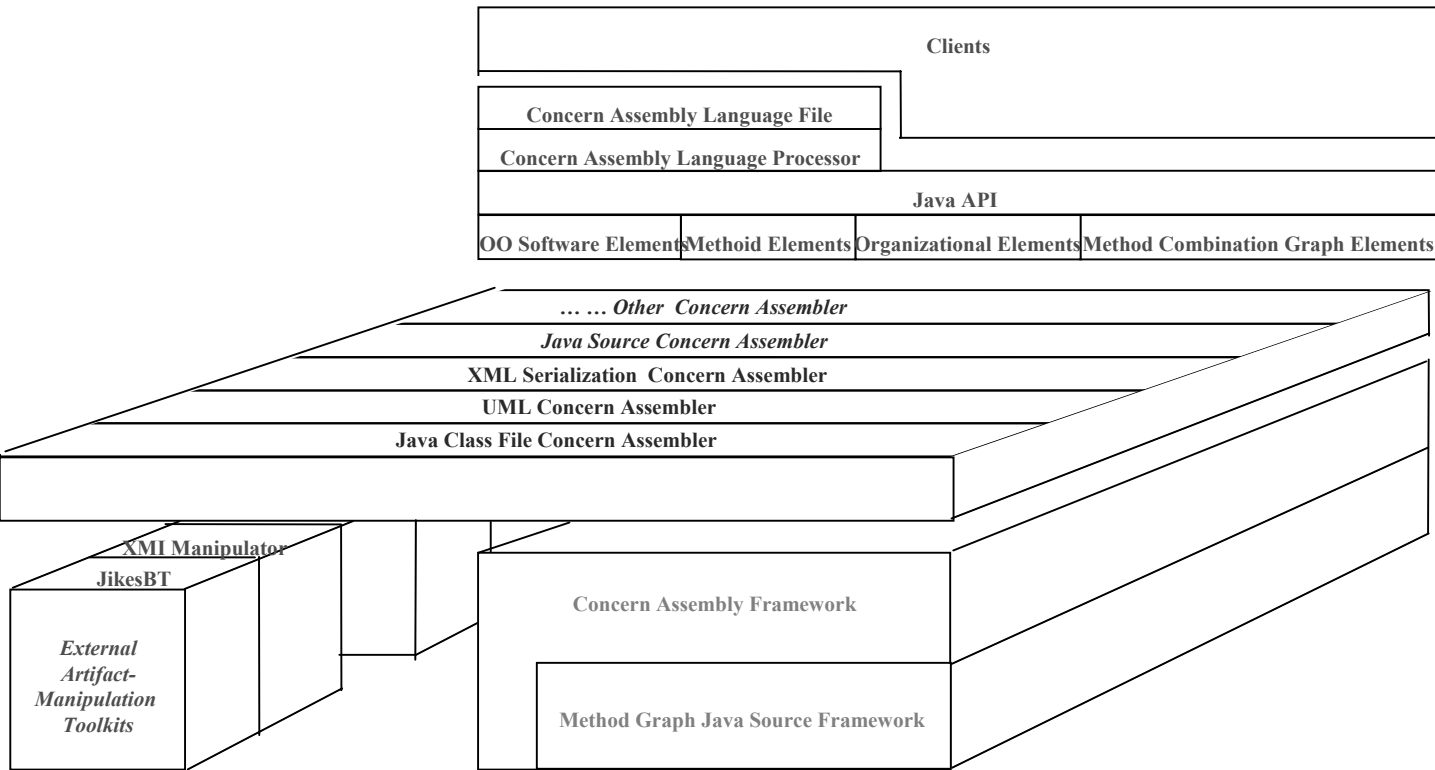
Two-tiered support for method combination graphs: general graph traversal for code generation, Java-specific source code generation

### Exploitation

The CAT framework has been used for building 6 concern assemblers by 2 different organizations at a cost of no more than 3 person-months each.

# Initial CME Frameworks

## CAT



Provides both an API and a concern assembly language (XML)

Implemented by a multiplicity of concern assemblers

Supported by:

- an externally available engine for OO representation, and
- a framework for additional CAT concepts, including Java source generation instance of method graph code generation

# Initial CME Frameworks

## CIT

### Important Concepts

Type (Class/Interface/Classifier), Method, Field – as usual in OO

Type Space – closed set of types

Universe – set of type spaces with a common representation

### API/Operational Protocol

Specify input types to be composed and expected for from library

Make method calls to extract information about:

- Inheritance relationships between types

- Members and types defined and accessible within types

- Modifiers, (keyword specified properties) of types and members

- Characteristics of types and members

### Extensibility/Flexibility Characteristics

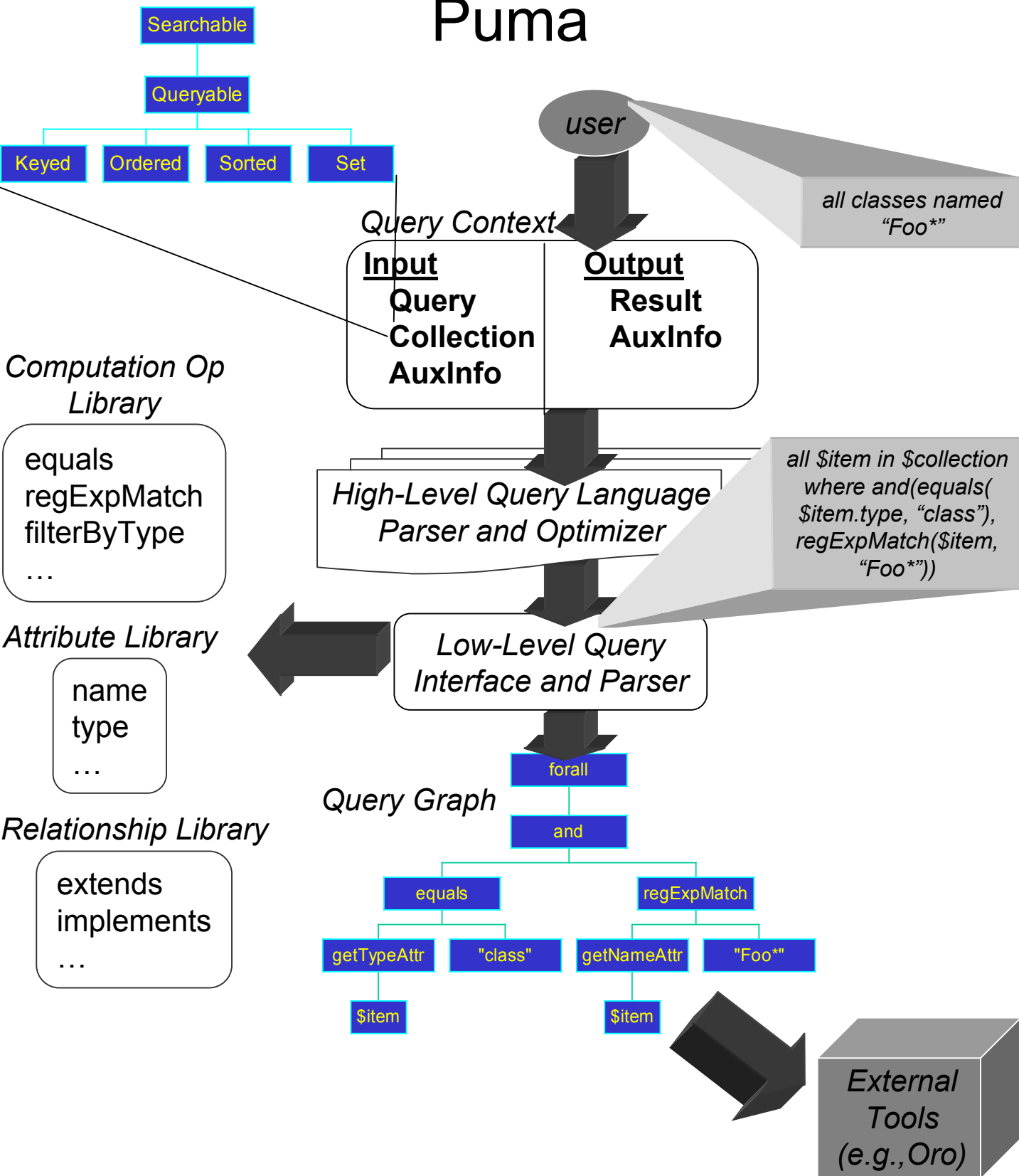
API – defined entirely as interfaces. Permits concern informants built as  
facades on top of manipulation engines

Independent sub-frameworks for universe, type space, method,  
method combination graph, type vector, modifiers

Sub-framework classes are to be subclassed for varied purposes, or can  
be used as are.

# Initial CME Frameworks

## Puma



# Initial CME Frameworks

## Puma

### API/Operational Protocol

- Create query context, which specifies collection of elements to be queried, the pattern sought, and any auxiliary information required
- Optionally specify desired result structure
- Run the query

### Important Concepts

**Collections:** The elements to be searched. Several different kinds of collections are distinguished (e.g., keyed, sorted), to aid in selecting appropriate queries.

**Patterns:** Strings written in some query notation that describe the characteristics of the elements sought.

**Auxiliary information:** Any information (aside from the collection) needed to execute the query, or helpful in executing it. (E.g., previously computed call graphs, index structures, etc.)

**Operator:** An executable unit that maps a collection to another collection.

### Extensibility/Flexibility Characteristics

- Open-ended set of operators, query languages and parsers, queryable elements, collection structures, search and optimization strategies

- User-specifiable result collection structure

- Cascaded queries

- Adapters for the Java collection interfaces (Map and Collection) provided

# Initial CME Engines

CAT/BT	Java binary file manipulator engine for CAT
CAT/JS	Java source file manipulator engine for CAT in Eclipse
CAT/XMI	XMI 1.0 file manipulator engine for CAT
CATCall	CAT couple interface for attaching many engines to one client
CATSerializer	CAT serializer interface for producing Concern Assembly Language (XML) files
CIT/BT	Java binary file manipulator engine for CIT