



SAFARI: A Meta-Tooling Platform for Creating Language-Specific IDEs

Robert M. Fuhrer, Philippe Charles, Stanley M. Sutton Jr.
(IBM T. J. Watson Research Center)

Chris Laffra
(IBM Rational)



Outline

- **Introduction**
- SAFARI IDE Development Process Walk-through
- SAFARI Architecture
- Status & Future Work

Motivation: Easier IDE Creation

- New programming languages are being developed all the time
 - “Pure” language research – X10, Fortress, SQLJ, XJ, Linq, PolyJ,...
 - Languages to support new architectures, environments, ...
 - Domain specific languages
 - Scripting languages
- Evaluation of language design requires analysis of prolonged use on significant code bases
- IDE support is **critical** to adoption and substantial use of new language
- Many existing languages still don’t enjoy support in mainstream IDEs

SAFARI Target: Desired IDE Functionality

syntax highlighting, compiler annotations,
hover help, source folding, formatting...

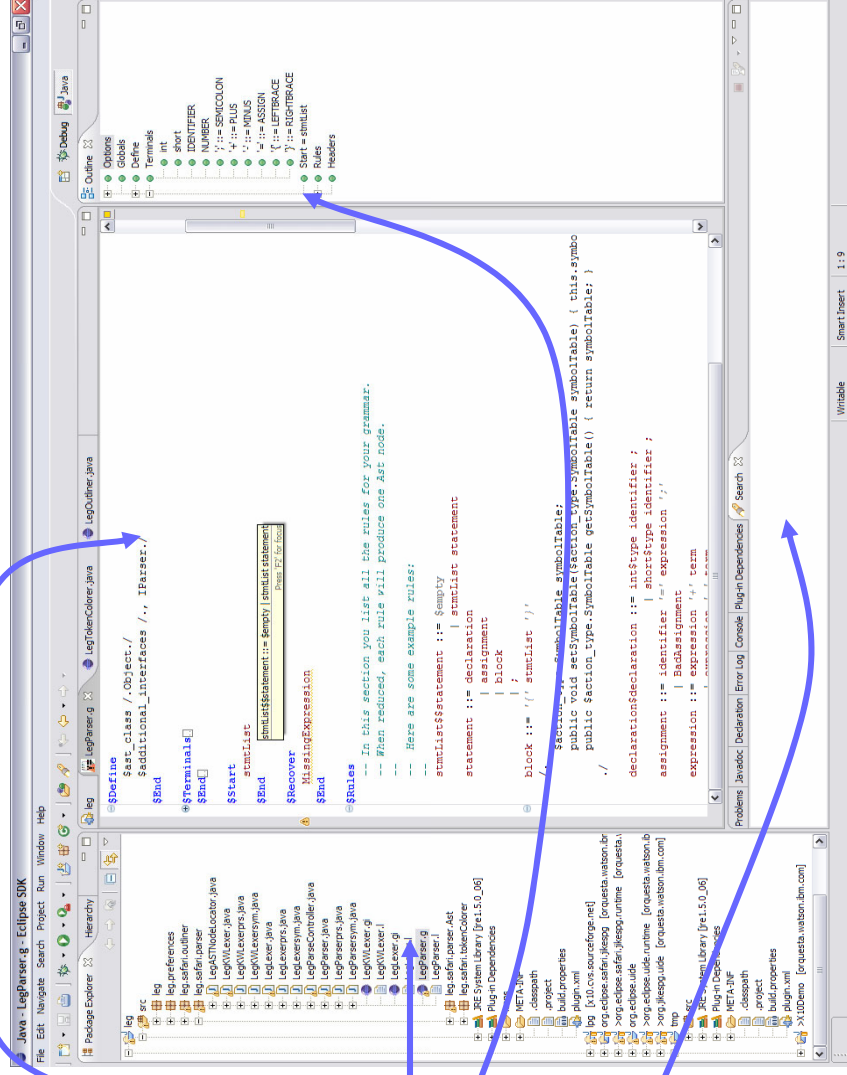
navigation (hyperlinks, “Open Type”, ...)

content assist, quick fixes

structural views

compiler w/ incremental build,
automatic dependency tracking

- New Project/Type/... creation wizards
- refactoring
- launch & debug: launch configs, breakpoints, backtraces, values, evaluation



JDT sets a very high bar!



SAFARI Approach

- Take advantage of common themes, structures, semantics
 - Encapsulate common IDE & language idioms
 - Language inheritance:
 - Δ in language structure/semantics $\Rightarrow \Delta$ in implementation
- Meta-tooling for language-specific IDEs
 - Language-definition support for syntax, auto-generated ASTs, analyses
 - Framework classes for IDE components
 - DSL's to more easily implement language services
 - Extensible multi-language static analysis framework (WALA)
 - Refactoring support
- Guide developer and direct focus to relevant APIs & customization sites
 - Cheat sheets, wizards, default implementations, example IDEs, ...

Enable IDE developers to get on with the interesting work!

Outline

- Introduction
- **SAFARI IDE Development Process Walk-through**
- SAFARI Architecture
- Status & Future Work

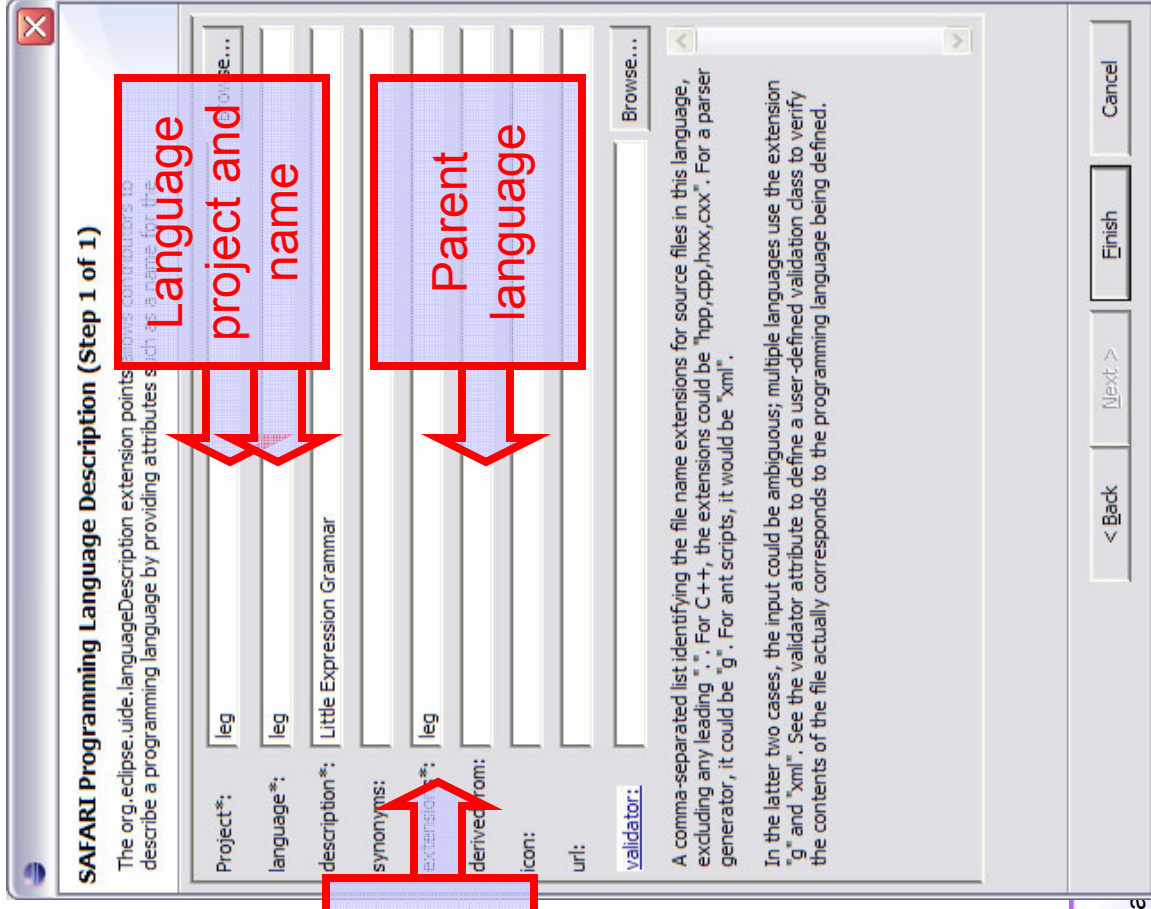
SAFARI Development Process: Overview

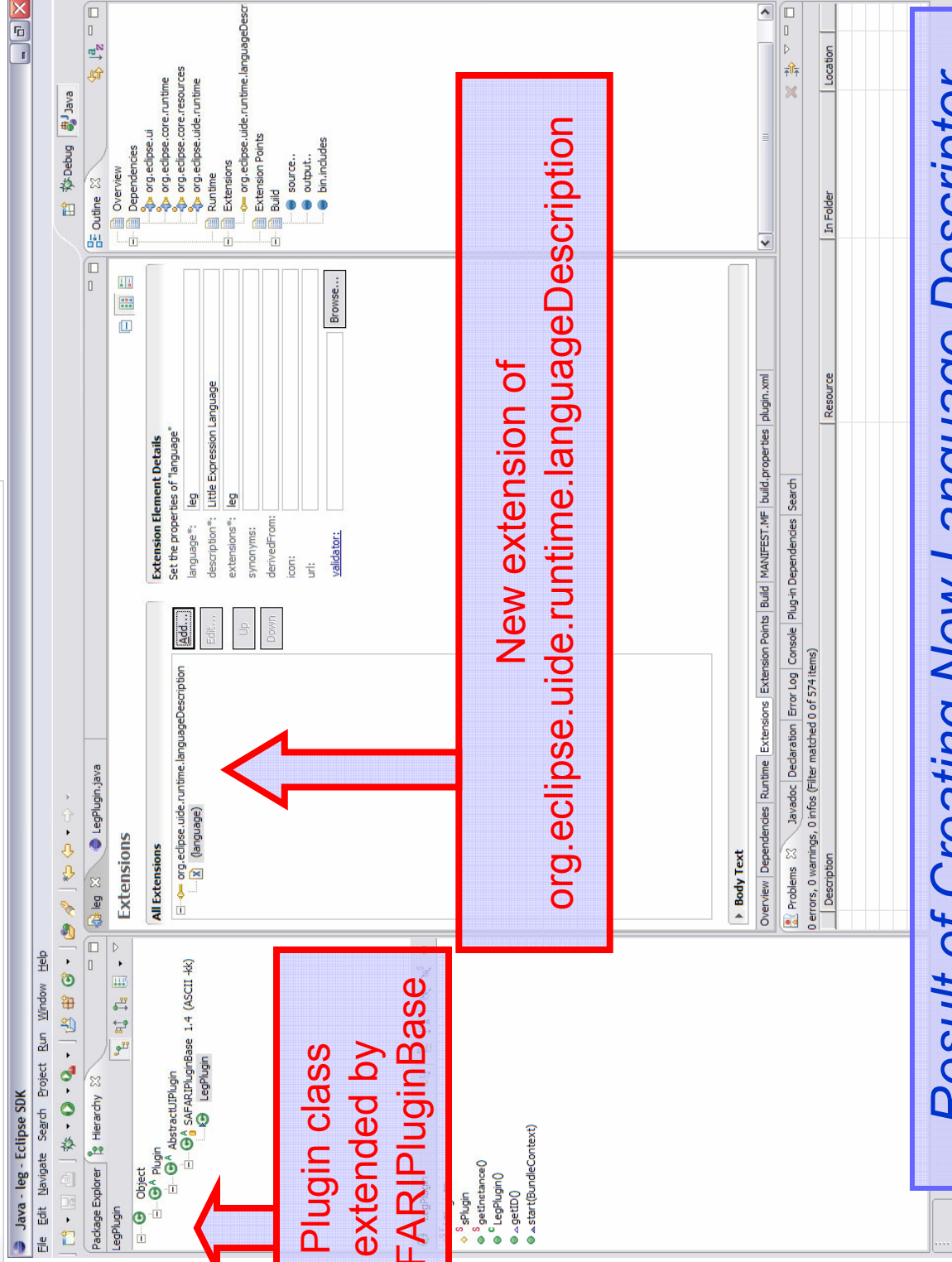
- Start with a plugin project
- Define language descriptor
 - Identify base language (if any), file name extensions, ...
 - In the future: use standard Eclipse “content types”
- Define lexical and grammar specifications
 - Using LPG: create grammar skeleton; complete it; parser and AST types automatically generated
 - In the future: interoperate with other parser generators
 - Or do it all yourself
- Define various language services
 - Mostly in any order, though a few constraints (e.g., reference resolver before content assistance)
 - Customize each selected service as necessary



Demo, part 1: Basic Services

Describe the New Language



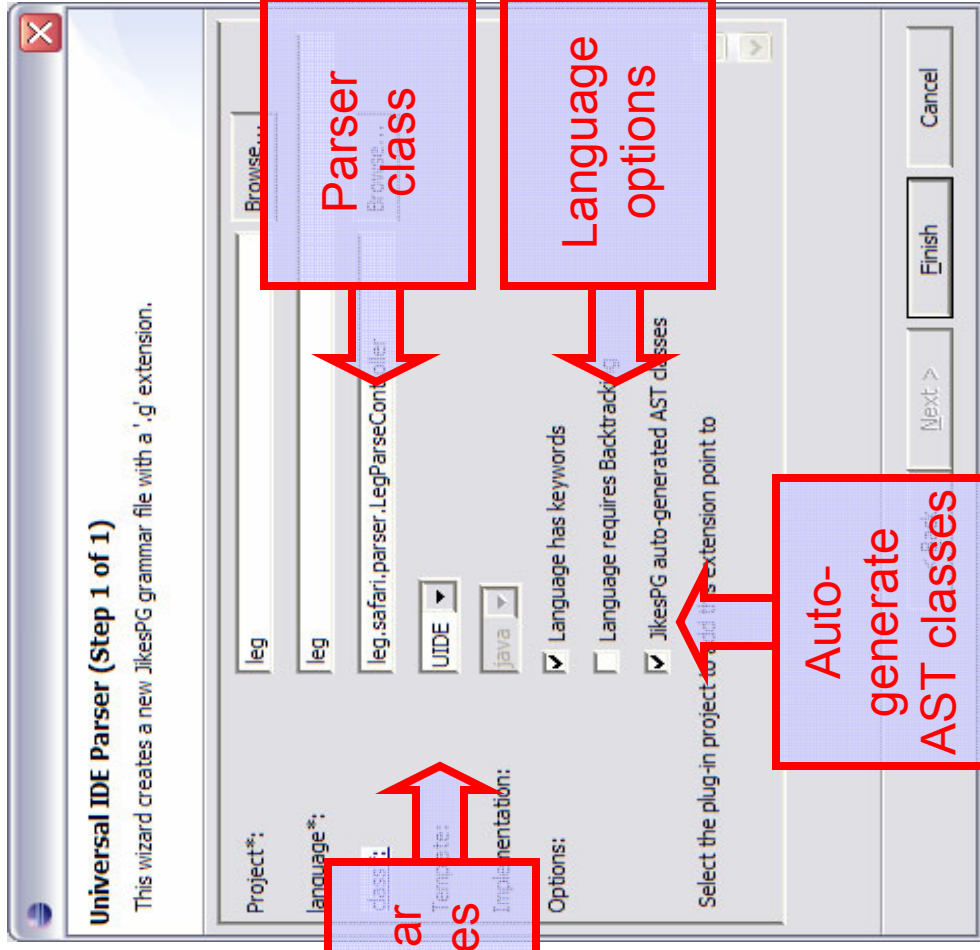
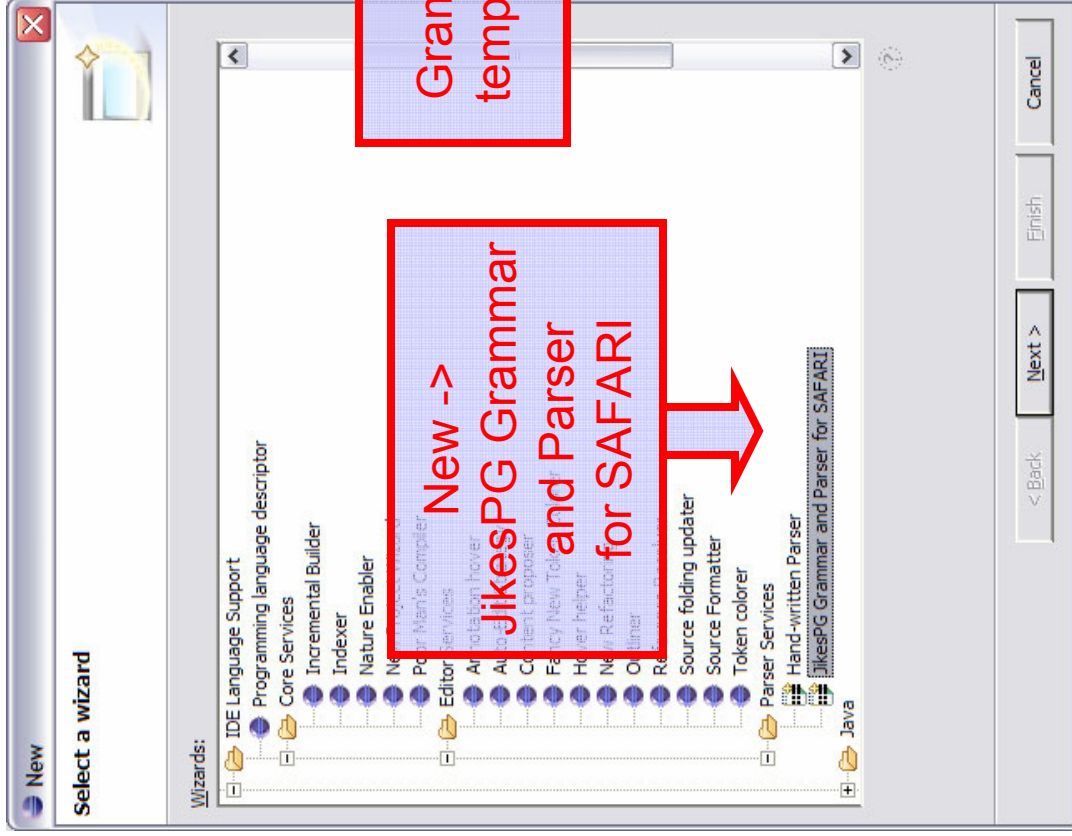


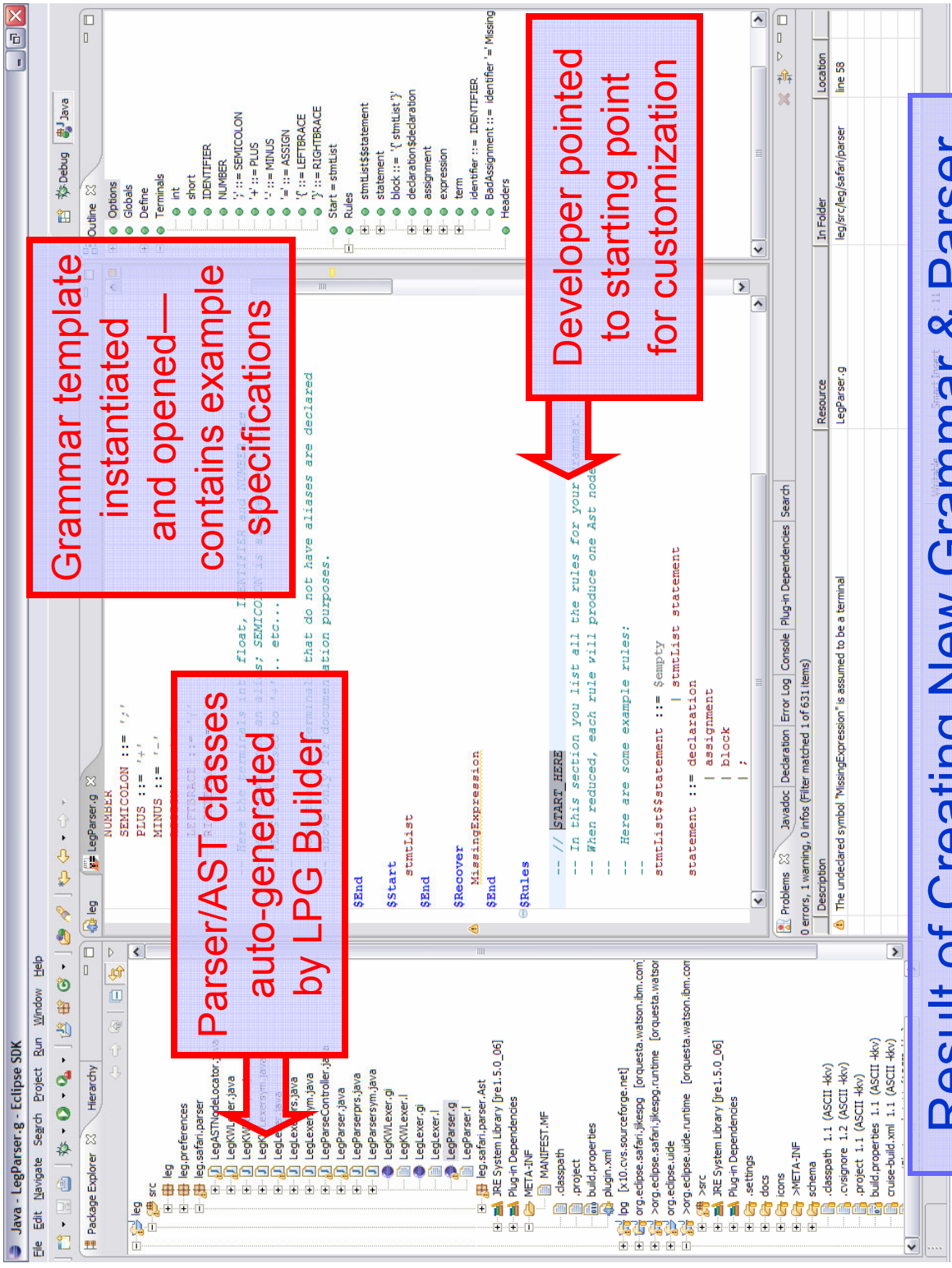
Plugin class extended by SAFARIPuginBase

New extension of org.eclipse.ui.runtime.languageDescription

Result of Creating New Language Descriptor

Create the Grammar & Parser



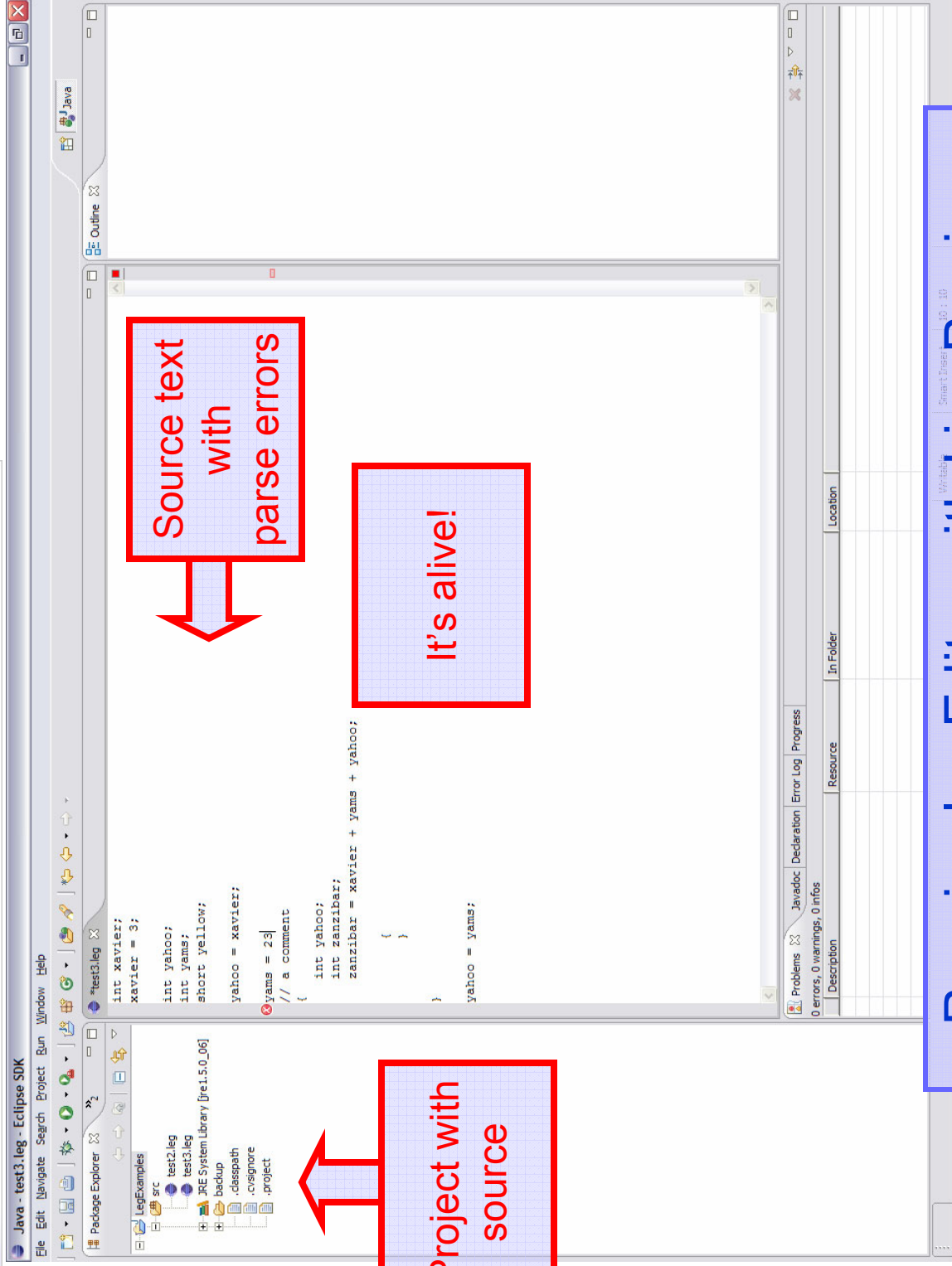


Grammar template instantiated and opened— contains example specifications

Parser/AST classes auto-generated by LPG Builder

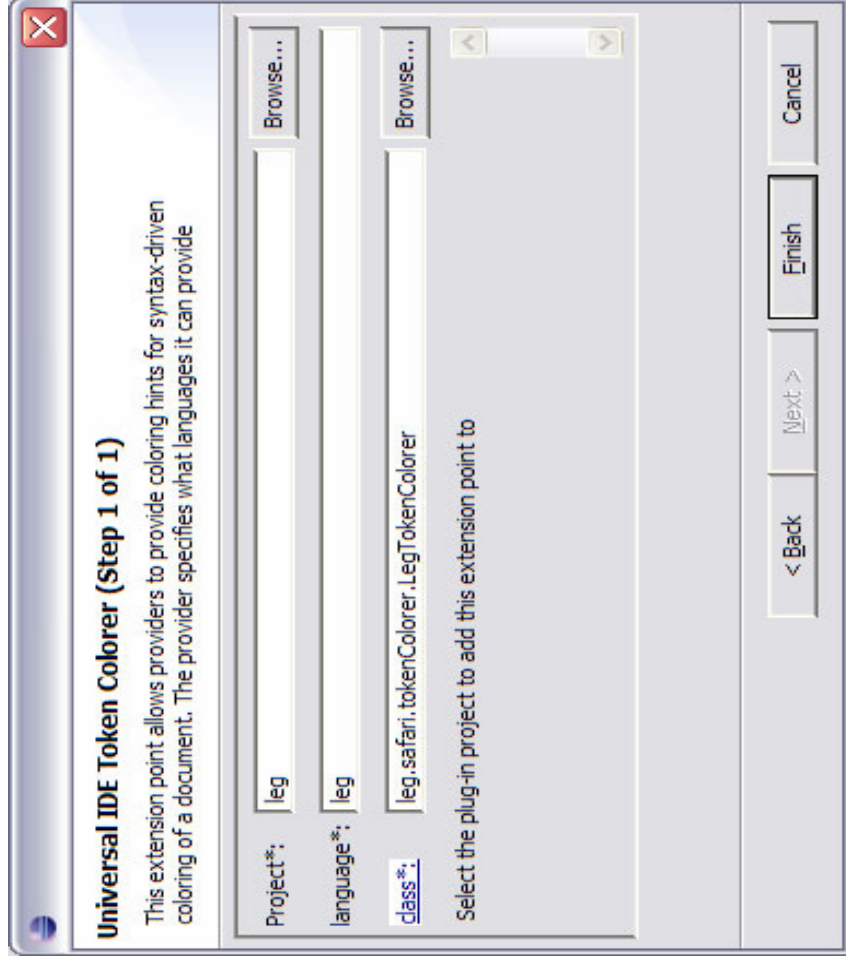
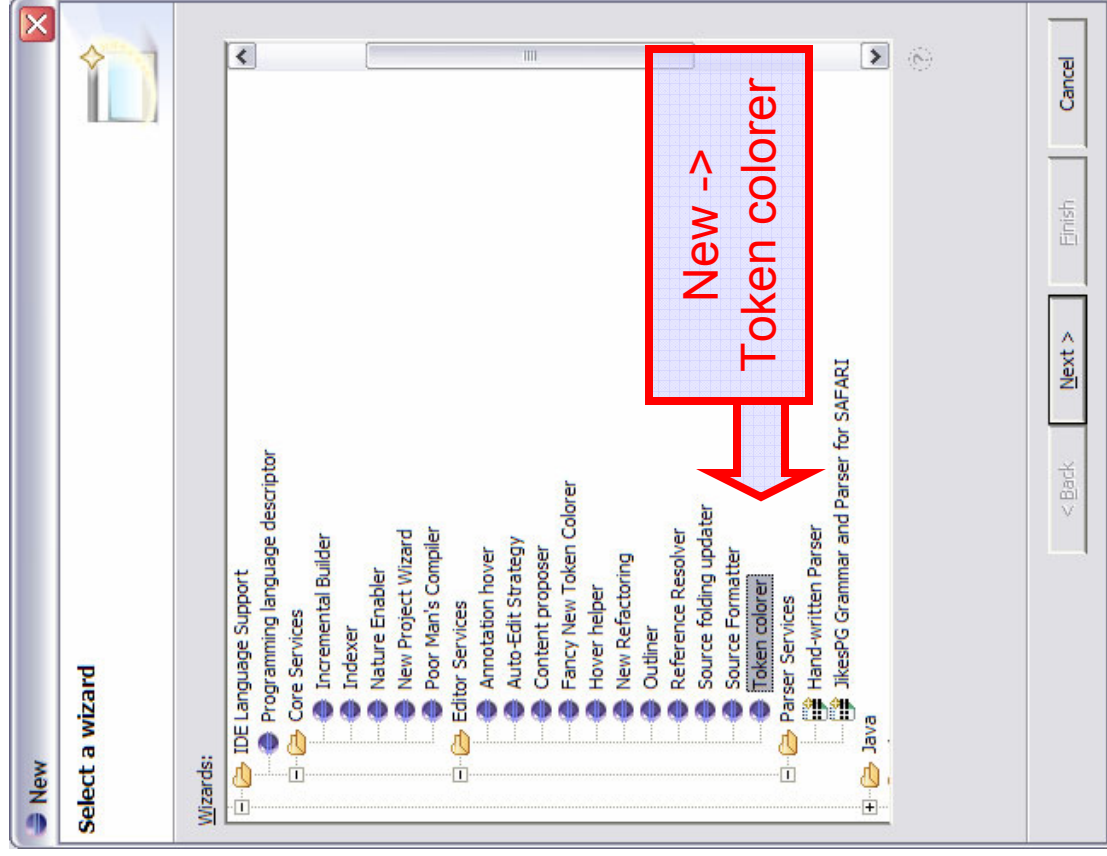
Developer pointed to starting point for customization for customization

Result of Creating New Grammar & Parser

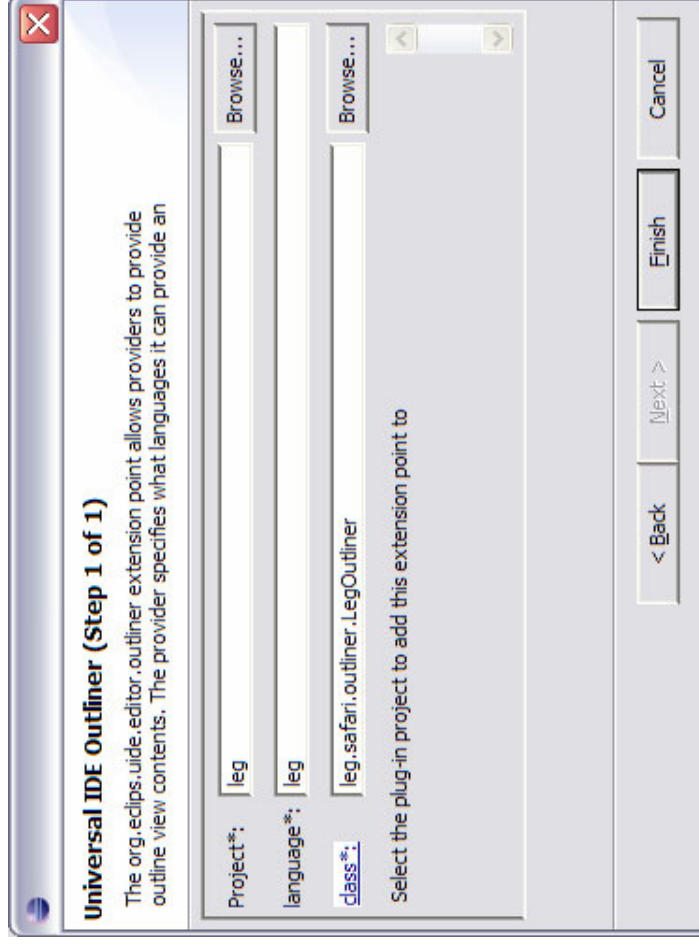
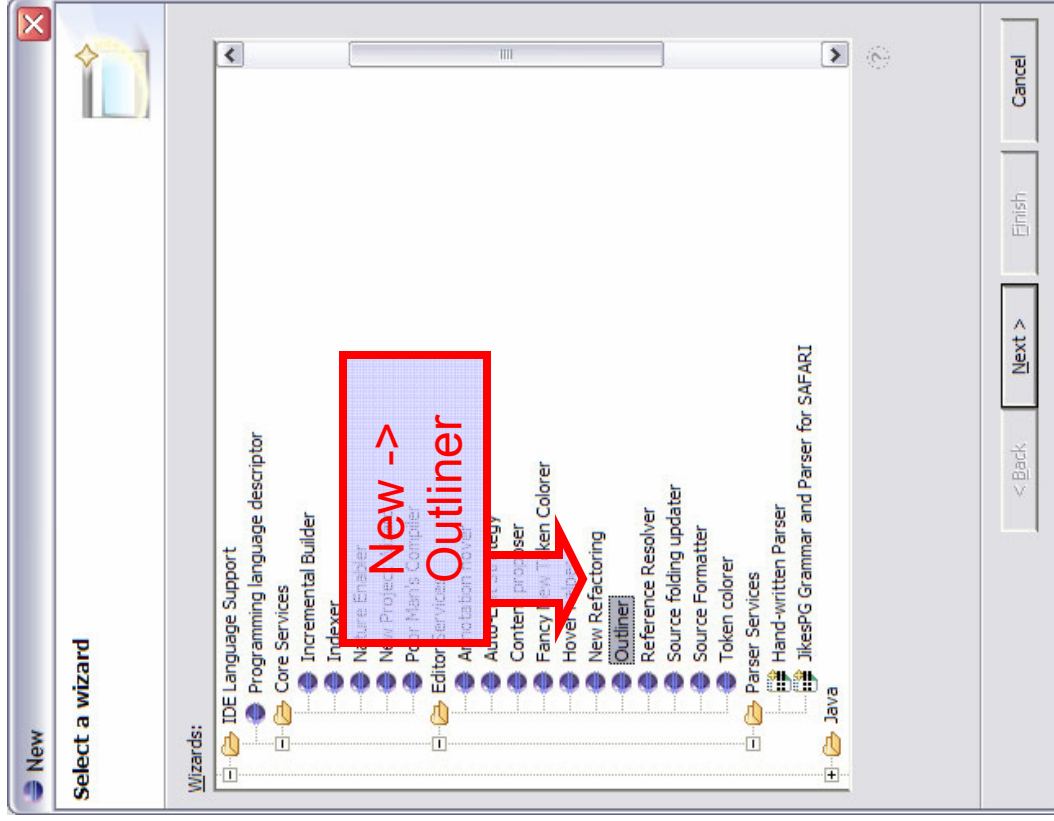


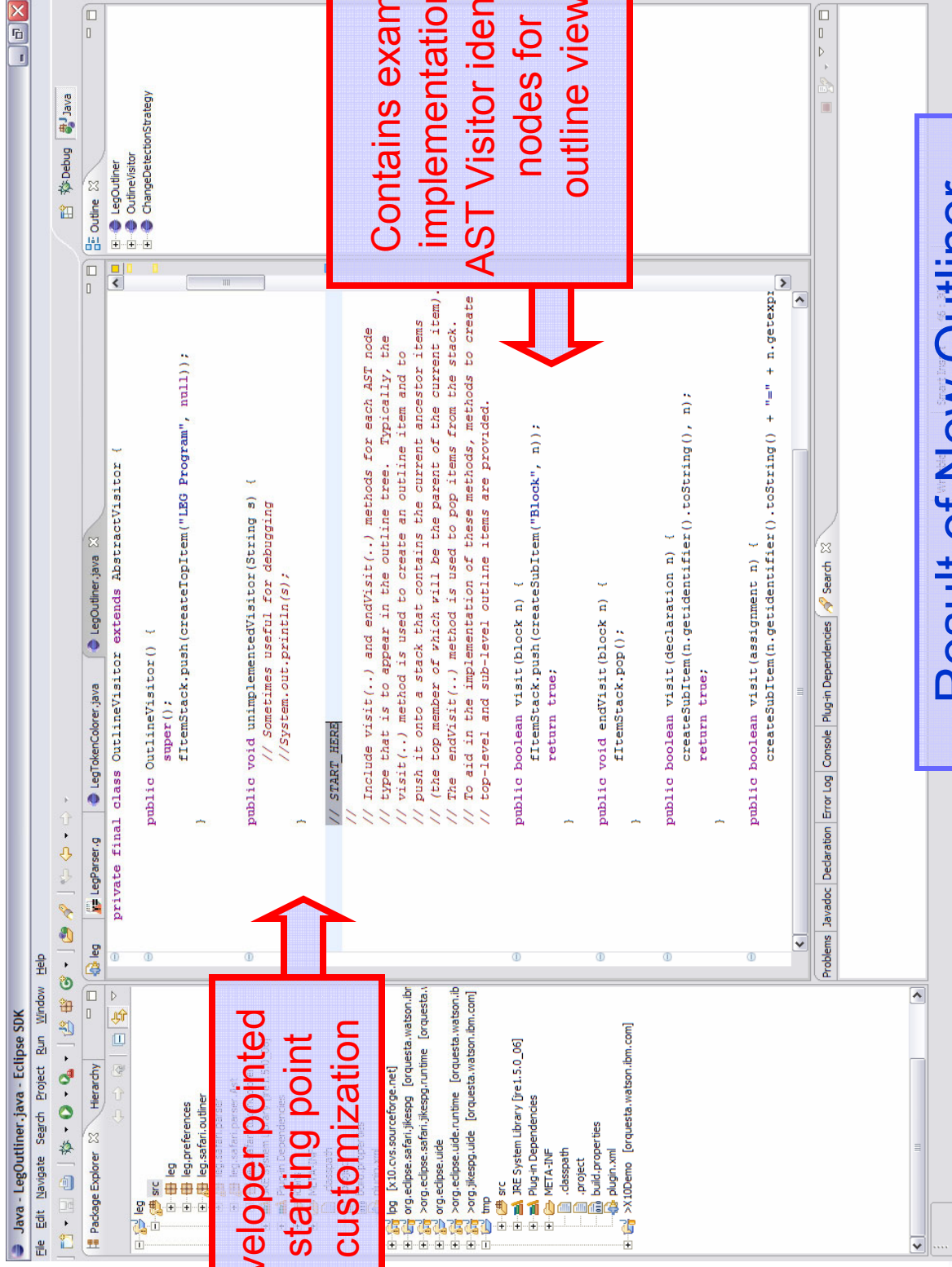
Running leg Editor with Live Parsing

Add a Language Service: Token Colorer



... Or Two: Outliner





Result of New Outliner

SAFARI Development Process: Adding a Builder/Compiler

- Create skeleton using wizard and SAFARI class library
- Flesh out skeleton:
 - Call out to an existing compiler
 - direct compiler messages to `IMessageHandler`
 - Write a new compiler starting from AST
 - If using Polyglot: implement standard analyses (type checking, reachability...)
 - Implement dependency visitor
- If compiler generates Java™ source: line breakpoint support by adding SMAP (JSR-44) attributes to generated Java class files
 - compiler inserts “`//Line`” comments to indicate original source location

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

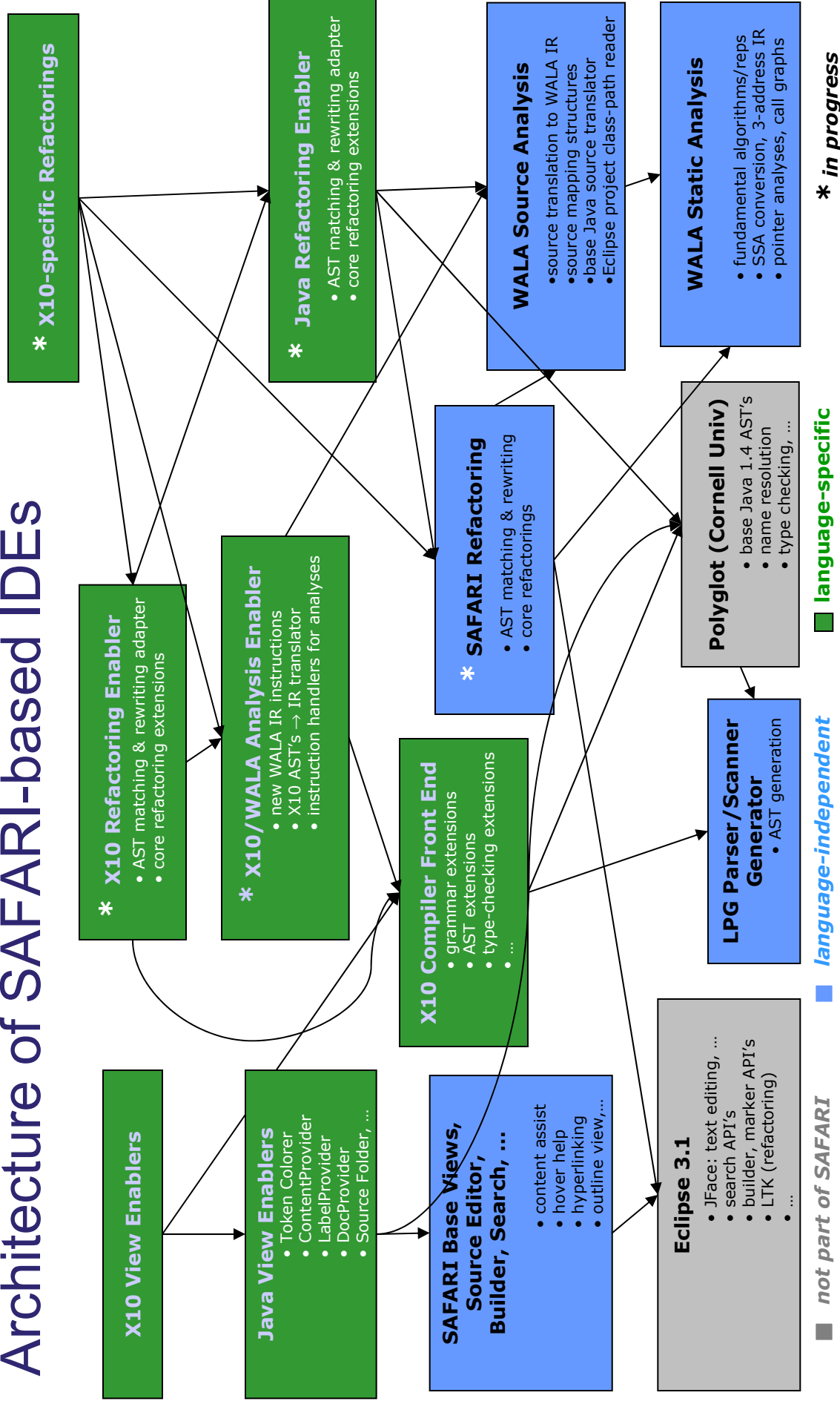


Demo, part 3: Building and Execution

Outline

- Introduction
- SAFARI IDE Development Process Walk-through
- [SAFARI Architecture](#)
- Status & Future Work

Architecture of SAFARI-based IDEs





SAFARI Support for Language Services & Analysis

Language Inheritance

Index Entry Specifications

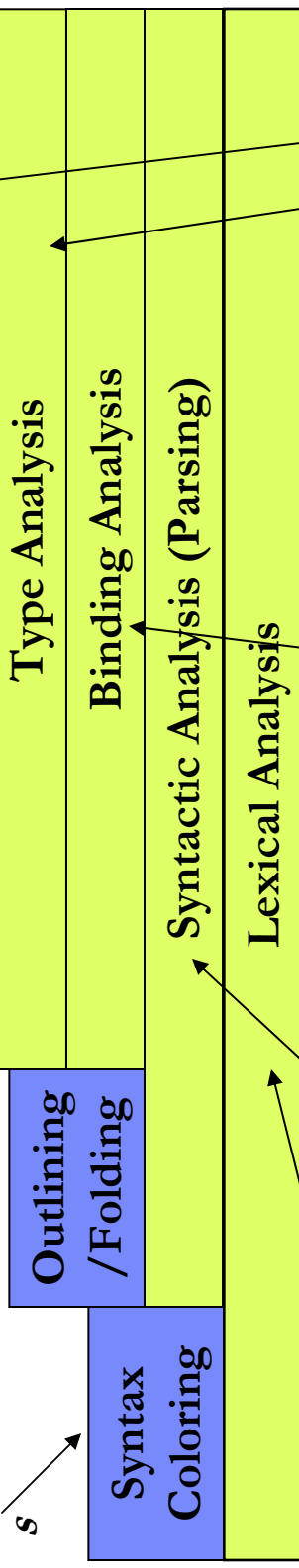
AST Pattern Matching

Declarative AST Rewrite Specifications

Error Recovery Grammar Specs, Declarative Assist

Declarative Specifications Presentation Specification

more expensive analyses



Lexical Specification

Grammar Specification, Auto-generated

n Scoping & Definitions

AST's

WALA Analyses

SAFARI (LPG) Scanner Specifications & Inheritance

Base Scanner:

```
ident ::= [a-zA-Z][a-zA-Z0-9_]+  
digit ::= [0-9]  
integer ::= digit+  
fixed ::= digit+ \. digit+  
...
```

Derived Scanner:

```
inherit base;  
drop float, fixed;
```

```
httpProto ::= 'http'  
mailProto ::= 'mailto'  
ftpProto ::= 'ftp'  
hostname ::= ident ( \. ident )+  
hostIP ::= digit+ ( \. digit+ )+
```

incremental additions
to base scanner spec

new terminals

SAFARI (LPG) Grammar Specifications & Inheritance

Base Grammar:

```
start A;  
A ::= B | C  
B $$b ::= b | B b  
mods [ | mod ] ::= $empty | mods mod  
mod## ::= static | public | ...  
...
```

AST for B is an array of b's

mods is an OR of mod's

AST for mod is an enum

Derived Grammar:

```
inherit base;  
drop C;
```

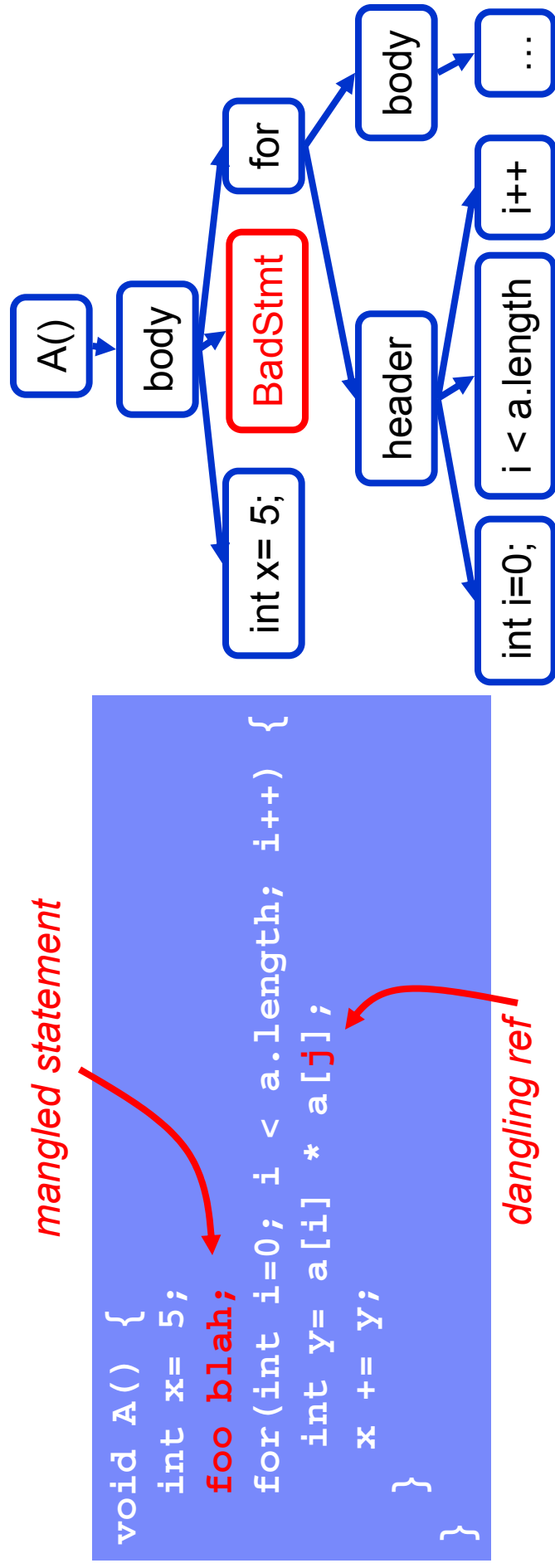
new production for
existing non-terminal

```
A ::= D  
D ::= ...
```

new non-terminal

Error Handling

- Errors are the norm! ⇒ must not cripple the IDE!
- All analyses must produce something reasonable wherever possible



- SAFARI/LPG: systematic, semi-automatic error recovery for parsing & creating “prosthetic” AST nodes

Outline

- Introduction
- SAFARI IDE Development Process Walk-through
- SAFARI Architecture
- **Status & Future Work**

Status and Future Work

- Implementation used @ IBM for ongoing IDE & language development
- Installation via IBM-internal Eclipse update site
- Current SAFARI-based IDE implementations:
 - LPG, Java, X10 (IBM Watson Research)
 - JavaScript (IBM Tokyo Research)
- Eclipse.org Technology Project proposal and initial open-source release planned for 2Q07
- Support for
 - Source formatting
 - Language embedding
 - Language inheritance
 - Refactoring and transformation
- Refinements and extensions to static analysis infrastructure

The End

Questions?

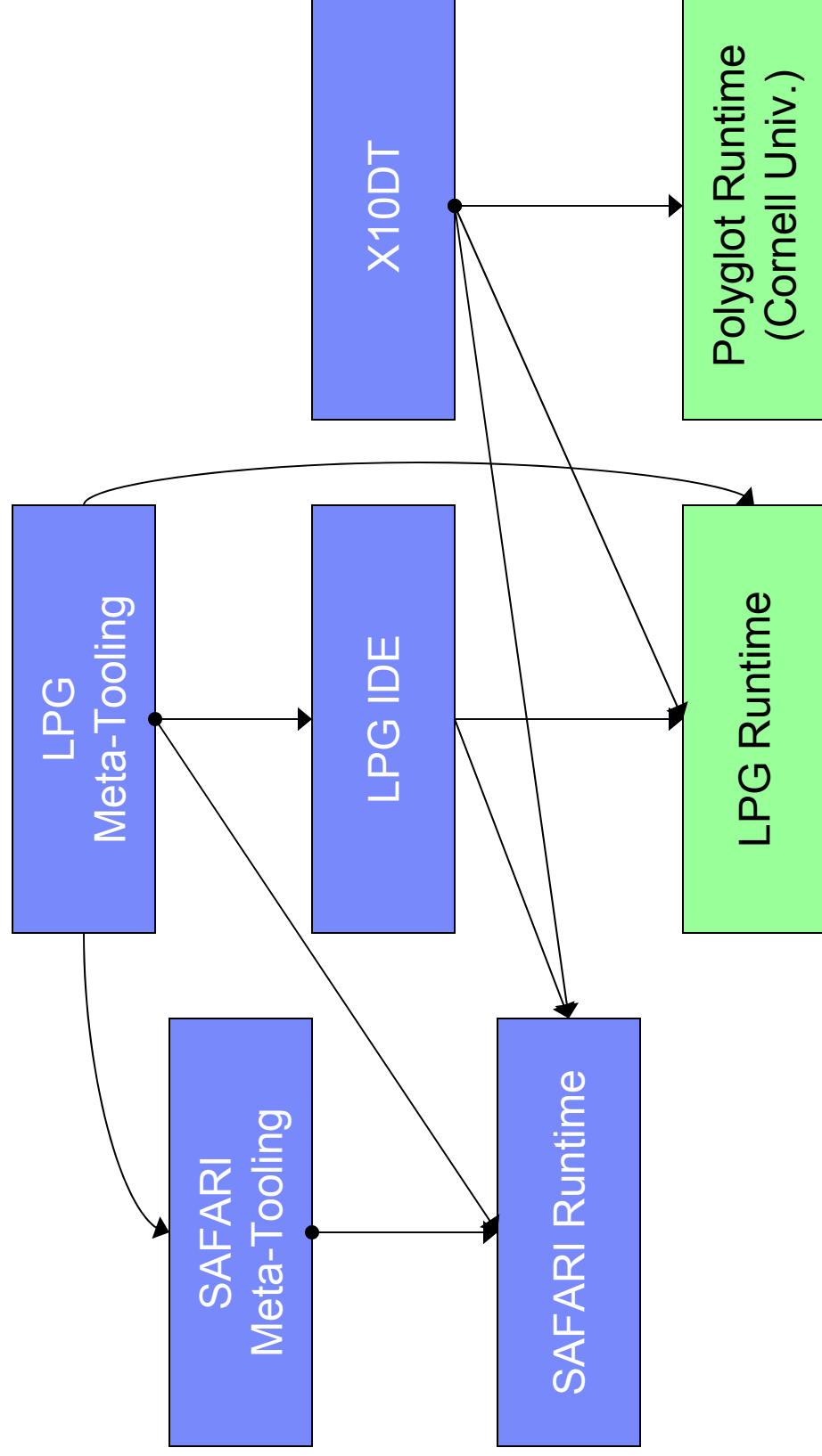
- SAFARI Meta-Tooling Platform
 - <http://www.research.ibm.com/safari/>
- LPG (formerly JikesPG) Scanner/Parser Generator
 - <http://sourceforge.net/project/lpg>
- The X10 Concurrent Programming Language
 - <http://x10.sourceforge.net/>
- WALA (formerly DOMO) Static Analysis Framework
 - <http://wala.sourceforge.net/>
- Polyglot Extensible Compiler Framework
 - <http://www.cs.cornell.edu/projects/polyglot/>

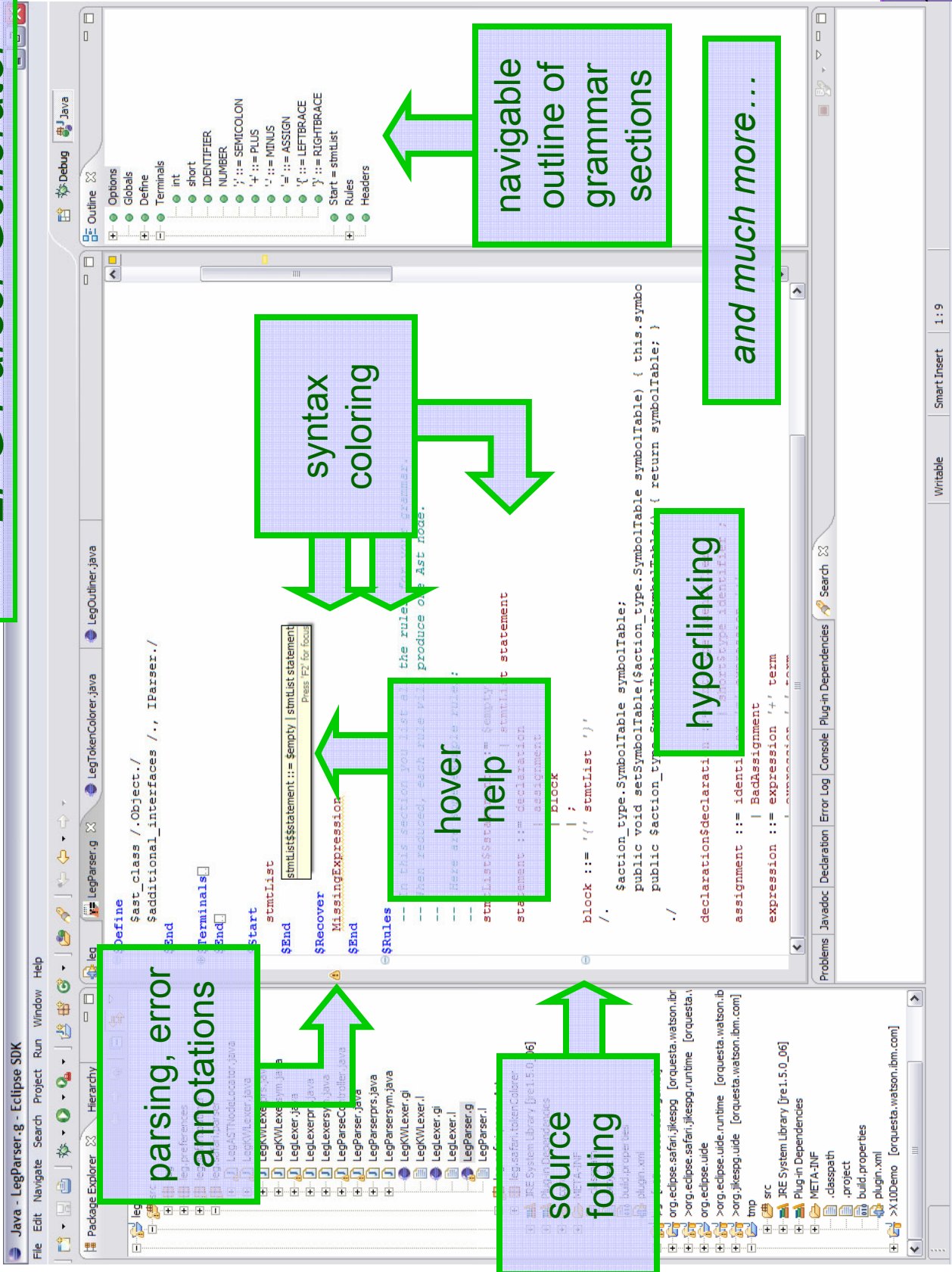
slides online here



Backup Slides

SAFARI Installable Features and their Dependencies







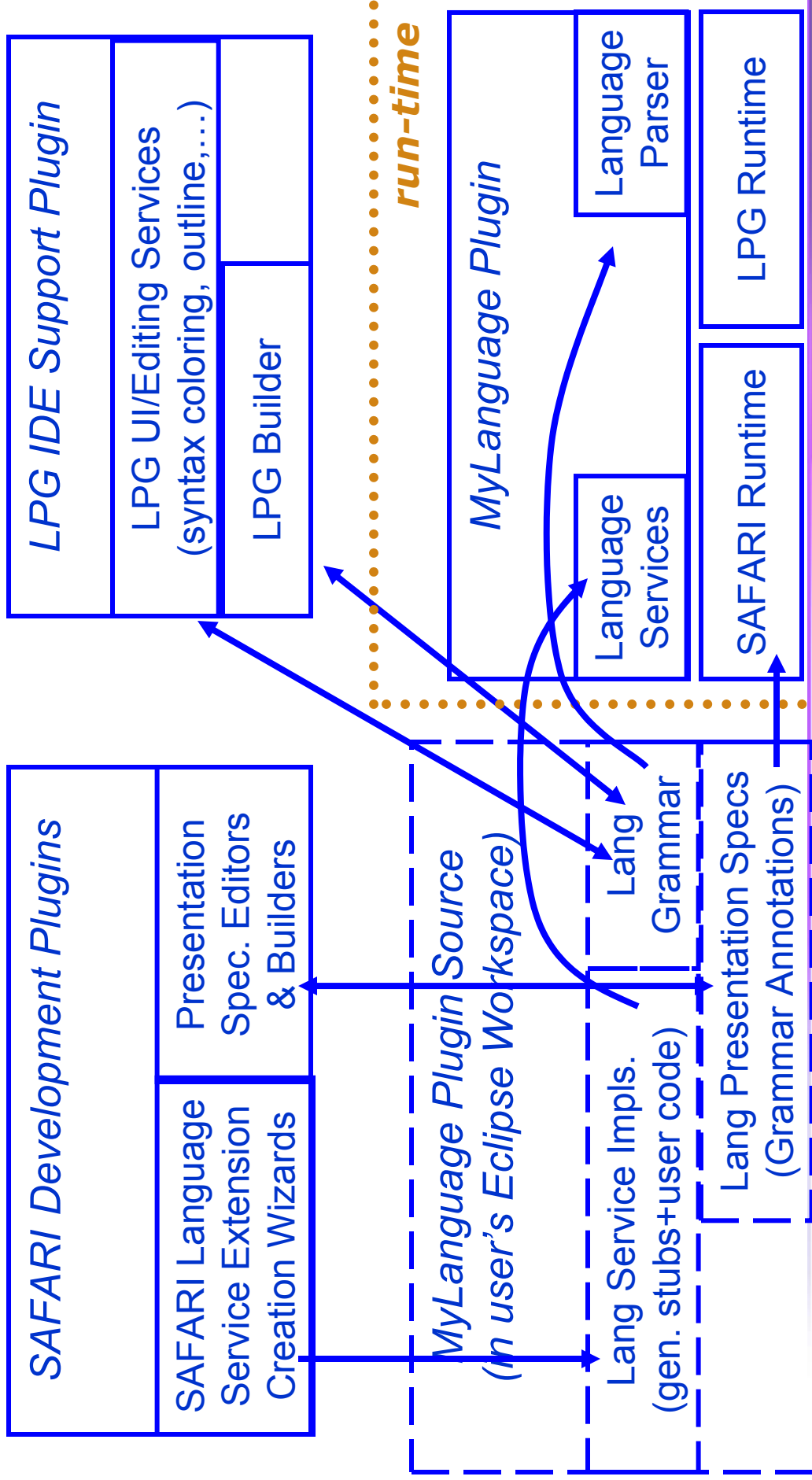
User-Visible IDE Services

IDE Developer Responsibilities

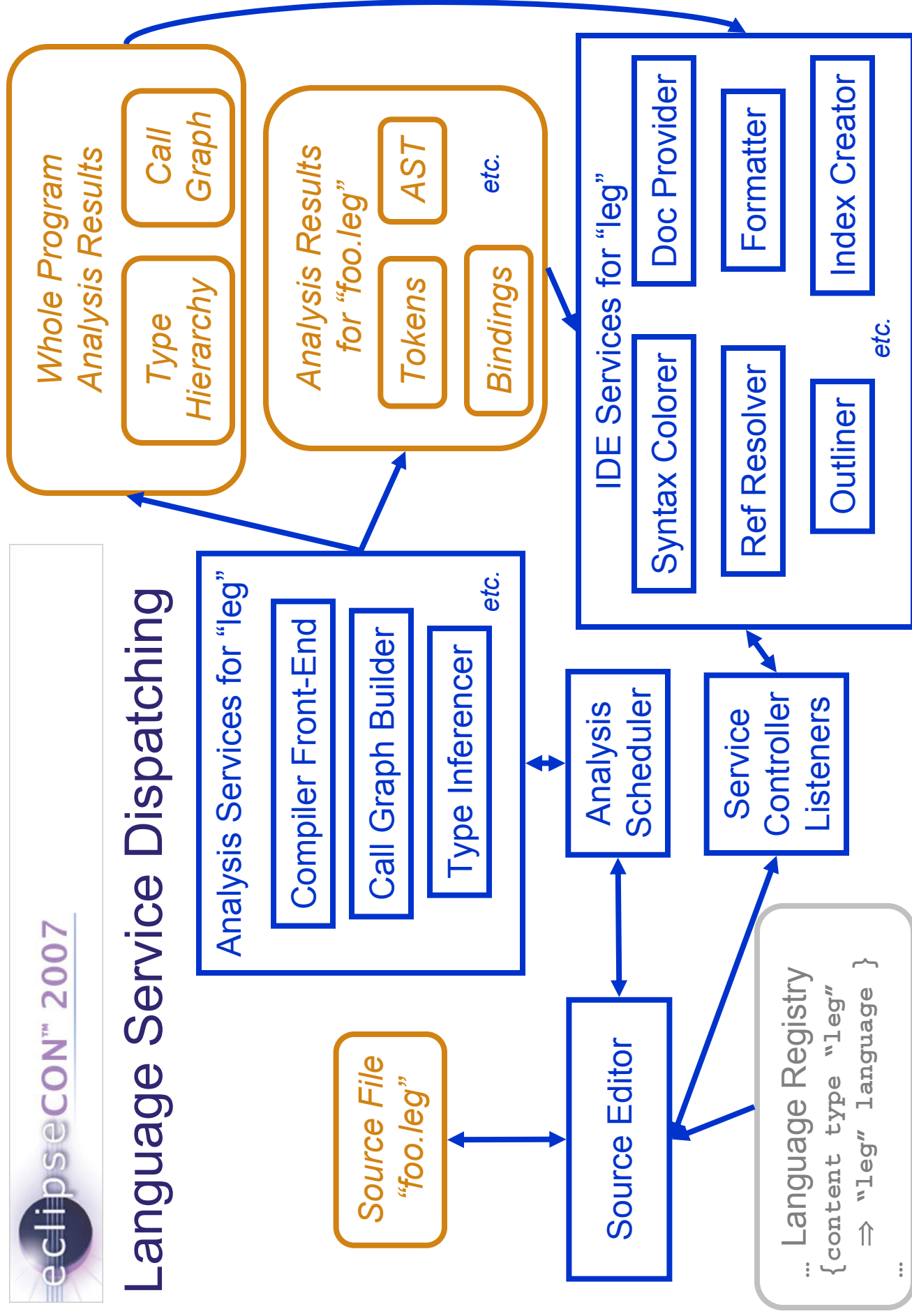
- Source editor
 - Compiler annotations
 - Annotation hover
 - Resource markers
 - Token coloring
 - Hyperlinked navigation
 - Hover help
 - Outline view
 - Quick outline
 - Content assistance
 - Indexed search
 - Source folding
 - Auto-editing
 - Formatting
 - Incremental compilation
 - Call graph
 - Type hierarchy
 - Refactoring contributions
 - Preference service and pages
- Language description
 - Parser; message handling
 - Token colorer
 - Reference resolver
 - Documentation provider
 - Outline content provider
 - Label provider
 - Image decorator
 - Content proposer
 - Index contributor
 - Folding updater
 - Auto-edit strategy
 - Formatter
 - Dependency scanner
 - Compiler
 - Nature enabler
 - Type analysis, IR construction
 - Refactoring contributions
 - Preference service and pages

SAFARI FLOW

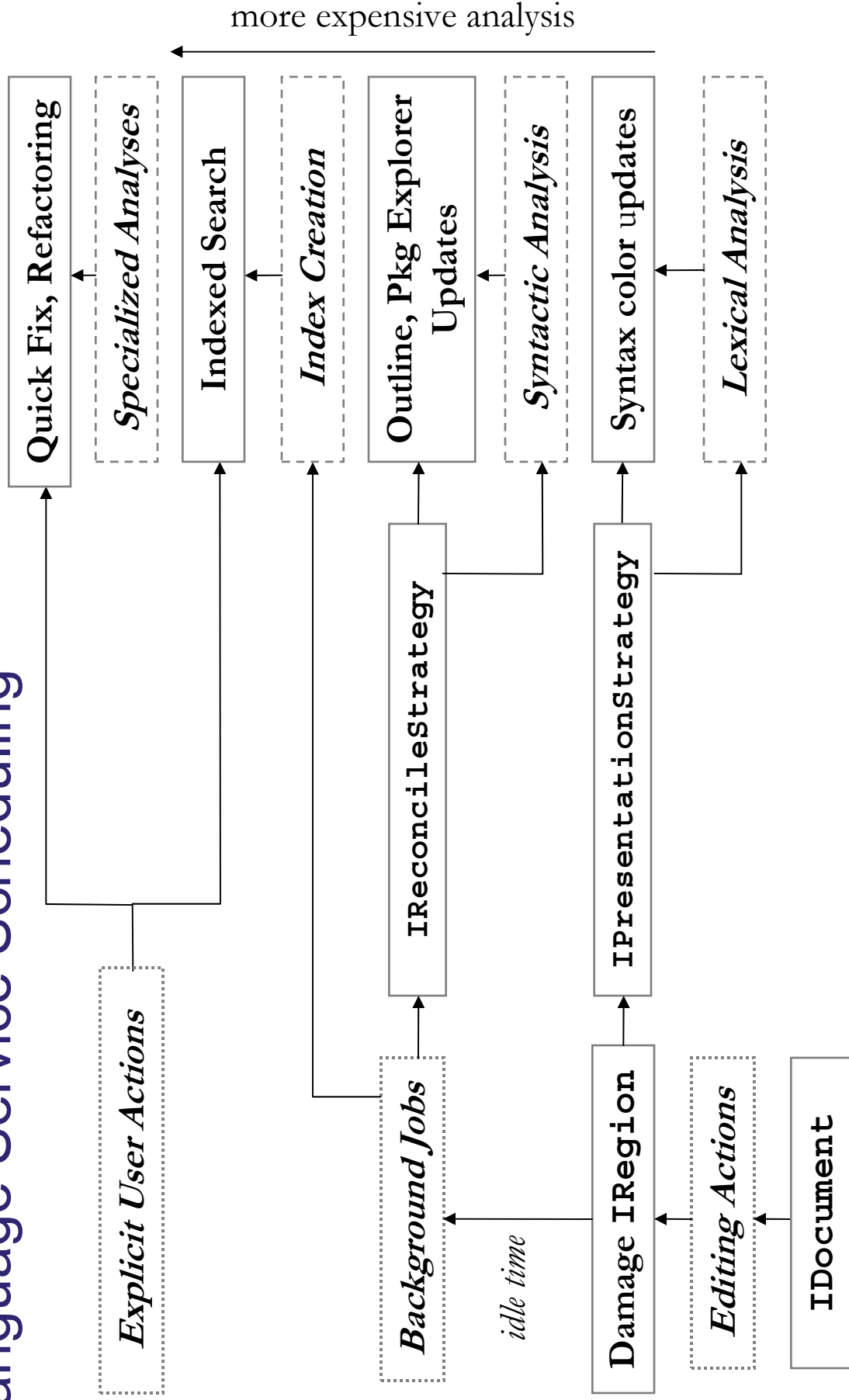
development time



Language Service Dispatching



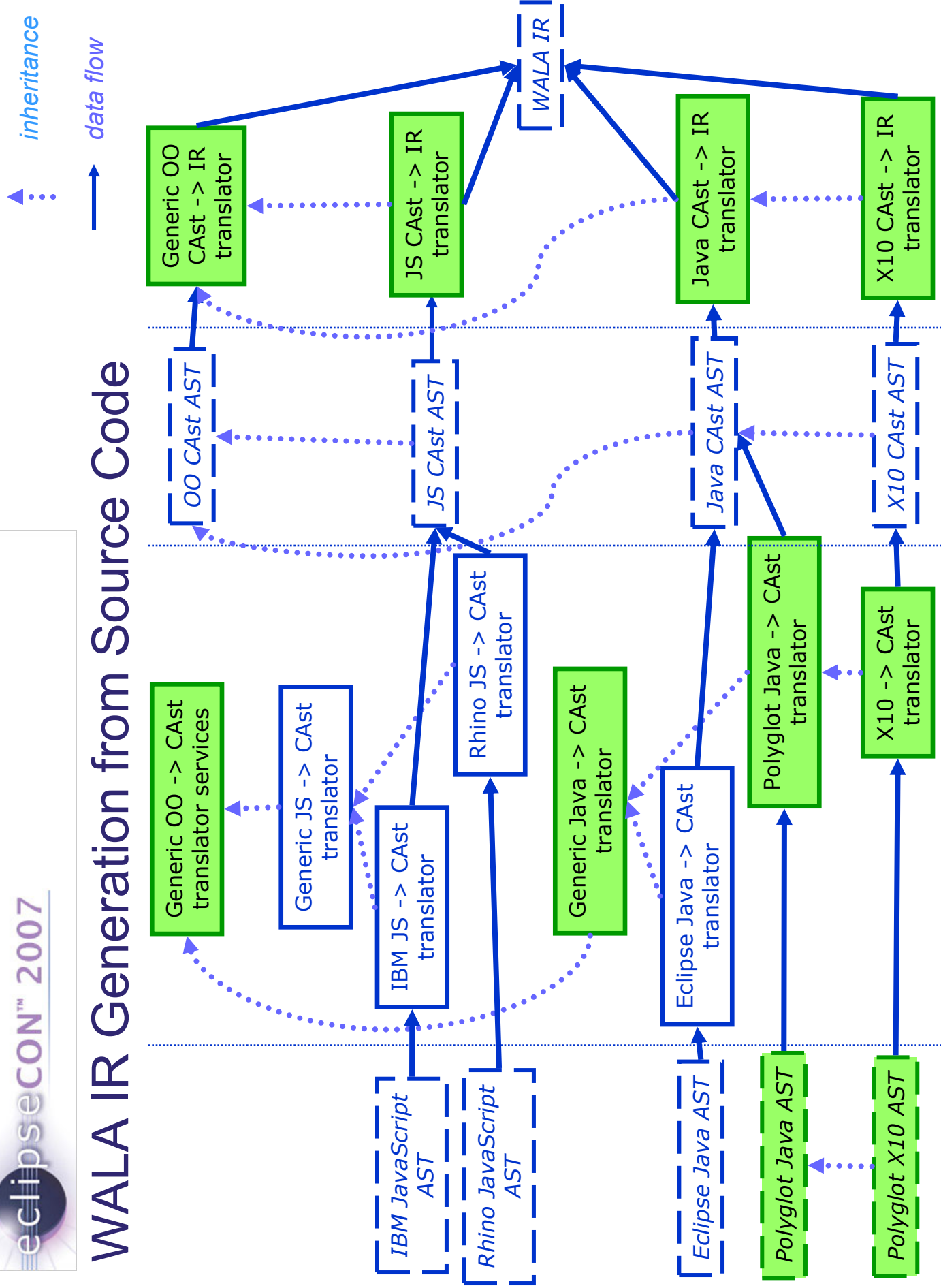
Language Service Scheduling



SAFARI Static Analysis Support

- Uses WALA open-source extensible static analysis framework
 - General framework encompassing many classic analyses
 - pointer, type, escape & effects analysis, call graph construction, ...
 - multiple precisions (CHA, RTA, 0-CFA, 1-CFA, etc.)
 - General iterative solver framework for expressing new analyses
 - Robust, highly scalable (capable of analyzing MLOC programs)
 - Handles static and dynamic languages
 - Currently supports Java, JavaScript, PHP, X10
- Adding support to WALA for a new language:
 - Implement translator from source AST's into WALA AST's
 - Define new instruction types for WALA IR as needed (~10 for X10)
 - Implement constraint handlers for new IR instructions to enable existing analyses (e.g. pointer analysis, effects analysis, escape analysis)

WALA IR Generation from Source Code





Related Work

- GUIDE (Laffra/IBM Rational):
 - inspiration, foundation for early SAFARI prototype
- Eclipse Language Development Toolkit (LDT):
 - Eclipse Technology Project proposal, vaguely similar goals to SAFARI, withdrawn
- Eclipse Web Standard Tools (WTP):
 - Focus on multi-language support
 - Structured Source Editor (SSE) offers similar editing infrastructure
 - API's, no meta-tooling (?)
 - May be possible to build parts of SAFARI on top of WTP/SSE (TBD)
- Eclipse Dynamic Languages Toolkit (Technology Project)
 - Focuses on dynamic languages
 - Uses single generic language model for program representation; SAFARI permits custom ASTs, and can use your existing compiler front-end as is
 - Not based on meta-tooling
 - Aims for language interoperability, SAFARI for IDE and language extensibility