


Architecture of EiffelStudio

Emmanuel Stapf
Eiffel Software
April 17th 2007


© 2007 Eiffel Software 1



Numbers from April 2006

- Command-line compiler only:
 - ~2100 classes (~460 for libraries)
 - ~440 000 lines of code (~120 000 for libraries)
- Full graphical IDE:
 - ~4200 classes (~1100 for libraries)
 - ~980 000 lines of code (~280 000 for libraries)
- C code:
 - ~100 000 lines of code


© 2007 Eiffel Software 4



Overview of EiffelStudio

- EiffelStudio by numbers
- General overview
- Compiler
 - Validation
 - Code generation
- User interface – EiffelStudio
- Repository
- Runtime
- Projects
- Q&A


© 2007 Eiffel Software 2



Overview 1 - EiffelStudio's architecture

- EiffelStudio contains:
 - Compiler
 - Debugger
 - Editor
 - Browsing tools
 - Reporting tools (warnings, errors, C compilation output)
- The graphical IDE contains the command line compiler.
- Command-line compiler can be compiled stand-alone.


© 2007 Eiffel Software 5



A few numbers

- Command-line compiler only:
 - 2548 classes (1083 for libraries)
 - ~1 250 000 lines of code (~935 000 for libraries)
- Full graphical IDE:
 - 5395 classes (2860 for libraries)
 - ~1 880 000 lines of code (~1 200 000 for libraries)
- C code:
 - ~100 000 lines of code

© 2007 Eiffel Software 3

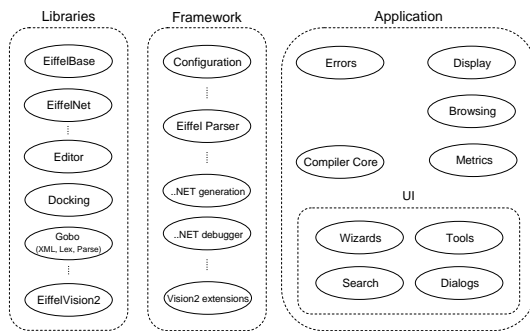


Overview 1 - EiffelStudio's architecture (2)

- At the source level, EiffelStudio uses:
 - Libraries
 - Frameworks
 - Its own code
- Framework is a library that is a reusable component but specialized for EiffelStudio. Some frameworks are good candidates for becoming libraries.

© 2007 Eiffel Software 6

Overview 1 - EiffelStudio's architecture (3)



© 2007 Eiffel Software

7

Patterns in EiffelStudio - Visitor



- Visitor pattern for traversing tree structures:
 - Compiler ASTs
 - Byte node ASTs (for code generation)
 - Types
- Initially visitor pattern was not used and some traversals are still done the old way (i.e. defining the same feature in all the descendant of a class).

© 2007 Eiffel Software

10

Overview 2 - Compilation process



- More at http://eiffelsoftware.origo.ethz.ch/index.php/Eiffel_Compilation_Explained
- Degree 6: finding classes
- Degree 5: parsing classes
- Degree 4: inheritance analysis
- Degree 3: type checking

© 2007 Eiffel Software

8

Patterns in EiffelStudio - Extensibility



- To abstract some platforms differences we use the extensibility pattern in:
 - Debugger (classic vs. dotnet)
 - Dotnet code generation (None/Microsoft .NET, and in the future Mono).

© 2007 Eiffel Software

11

Overview 2 - Compilation process (2)



- Degree 2/1: melting
- Degree -1: freezing
- Degree -2,-3: finalization
 - Degree -2: process polymorphism
 - DCR: Dead Code Removal
 - Degree -3: code generation

© 2007 Eiffel Software

9

Patterns in EiffelStudio - Factory



- Factories are either used for abstracting:
 - Platform specific implementation (same purpose as extensibility)
 - Different needs in functionality:
 - See AST_FACTORY descendants
 - See CONF_FACTORY descendants

© 2007 Eiffel Software

12

Patterns in EiffelStudio - Others



- Other patterns in use in EiffelStudio:
 - Observer
 - Singleton
 - Lazy initialization
 - Flyweight

© 2007 Eiffel Software

13

Compiler - Types



- All types appearing in an AST are transformed into instances of *TYPE_A*.
- *TYPE_A* descendants:
 - *CL_TYPE_A*
 - *GEN_TYPE_A*
 - *TUPLE_TYPE_A*
 - *LIKE_FEATURE*
 - *FORMAL_A*
 - ...

© 2007 Eiffel Software

16

Compiler - AST



- All classes representing *AST* nodes are descendants of *AST_EIFFEL* and have the *_AS* suffix.
- Parser written using gelex/ge yacc.
- Parser has many faces:
 - Syntax checker: no AST, useful for syntax validation.
 - Light parser: keeps only nodes needed for validation.
 - Full parser (aka roundtrip parser): preserves all information about Eiffel text (code, blanks and comments).

© 2007 Eiffel Software

14

Compiler - Features



- The features of a class are stored in *CLASS_C* into an instance of *FEATURE_TABLE*.
- A *FEATURE_TABLE* is a container of *FEATURE_I*, indexed by feature names and, for fast lookup, by "routine IDs".
- Descendants of *FEATURE_I*:
 - *PROCEDURE_I*
 - *DYN_FUN_I*
 - *ATTRIBUTE_I*
 - *EXTERNAL_I*
 - ...

© 2007 Eiffel Software

17

Compiler - Classes



- Every class has an associated *CLASS_I* instance.
- *CLASS_I* stores information about the file holding the class text: modification date, class name, associated cluster.
- Classes that are part of the system also have an associated *CLASS_C* instance.
- *CLASS_C* stores relations between classes as well as its features.

© 2007 Eiffel Software

15

Compiler - IDs

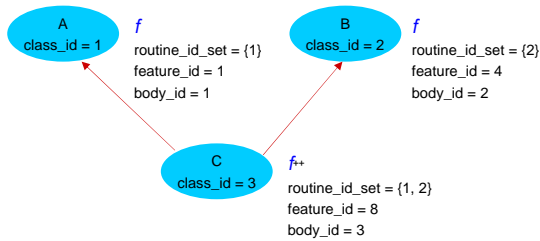


- Class ID: identifier given to each class.
- Routine ID: identifier given to each feature globally for polymorphism
- Feature ID: identifier given to each feature within a class
- Body ID (aka Body Index): identifier given to a feature text

© 2007 Eiffel Software

18

Compiler - IDs



© 2007 Eiffel Software

19

EiffelStudio - Editor



- Designed as a library.
- Configured by EiffelStudio to add:
 - Code completion
 - Pick and drop/Context Menu
 - Syntax highlighting
- Used for displaying code, but also results of formatters (views: flat, contract, interface...)
- **TEXT_PANEL** is the ancestor to all editors

© 2007 Eiffel Software

22

Compiler - Code Generation



- At degree 3 each feature is transformed into a **BYTE_CODE** instance, a tree of **BYTE_NODES**.
- Different types of code generation:
 - Melting
 - Freezing
 - Finalization
 - .NET freezing
 - .NET finalizing
 - Java freezing
 - Java finalizing

© 2007 Eiffel Software

20

EiffelStudio - Tools



- Controlled by **EB_TOOL**
- Information outputs:
 - Compilation global process, system information
 - Errors
 - Warnings
 - C compiler output
- Executing commands from EiffelStudio: svn status, svn update, svn commit...

© 2007 Eiffel Software

23

Dynamic dispatch



- Based on routine IDs
- Each routine ID is associated with a virtual table indexed by the dynamic type of an object at runtime.
- Generated code looks like:
a.f (args) ⇔ routine [dynamic_type (a)] (args)

© 2007 Eiffel Software

21

EiffelStudio - Diagram tool



- Uses graph library as data structure for internal representation:
 - Inherits from **EG_NODE**
 - Supports "physics" (force directed layout)
- Drawing done using model cluster of EiffelVision2 (**EV_MODEL_WORLD**)
- Two models are supported:
 - BON (**BON_CLASS_DIAGRAM**)
 - UML (UML subset, **UML_CLASS_DIAGRAM**)

© 2007 Eiffel Software

24

EiffelStudio - Queries



- Unification of classes/features/metrics facilities through a query language
- Grammar not fully specified yet
- What we have in mind: something like

```
select classes
from cluster=base
where count(features) > 10
```
- Work still in progress

© 2007 Eiffel Software

25

EiffelStudio - Navigation (3)



- **STONE** descendants:
 - **CLASSI_STONE**: non-compiled class
 - **CLASSC_STONE**: compiled class
 - **CLUSTER_STONE**: cluster/group/library/assembly
 - **FEATURE_STONE**: feature in context of a class
 - **ERROR_STONE**: compilation error
 - **OBJECT_STONE**: object in debugger
 - ...

© 2007 Eiffel Software

28

EiffelStudio - Navigation



- Search facility (**EB_MULTI_SEARCH_TOOL**):
 - Multiple scope: class, cluster, multiple clusters, system
 - Regular expression support
 - Search bar add-on to all editors
- Clusters and classes: **EB_CLUSTER_TOOL** and **EB_CLASSES_TREE**
- Features tree: **EB_FEATURES_TOOL** and **EB_FEATURES_TREE**

© 2007 Eiffel Software

26

EiffelStudio - Navigation (4)



- Locate a class or feature through an instance of **EB_ADDRESS_MANAGER**
- Used under two forms:
 - As toolbar
 - As modal dialog from context tool
- But same semantics

© 2007 Eiffel Software

29

EiffelStudio - Navigation (2)



- Pebbles used for Pick and Drop are descendants of **STONE**: **CLASSI_STONE**, **CLASSC_STONE**, ...
- Communication between all graphical elements is done through a stone (instance of **STONE**)

© 2007 Eiffel Software

27

EiffelStudio - Main window



- **EB_DEVELOPMENT_WINDOW**
 - Top level window in EiffelStudio
 - Handles all tools (clusters, features, context tool, editor, search,...) and their layout
 - Handles tool synchronization through stones
 - Handles creation of menus and commands
 - Two state: developing or debugging

© 2007 Eiffel Software

30

Adding New Tool in EiffelStudio



- A good tutorial can be found at:
http://eiffelsoftware.origo.ethz.ch/index.php/How_to_add_a_tool_to_Eiffel_Studio
- Summary:
 - Add class which inherits from `EB_TOOL` and implements the deferred features
 - Add tool to `EB_DEVELOPMENT_WINDOW_TOOLS`
 - Add tool creation to `EB_DEVELOPMENT_WINDOW_MAIN_BUILDER`

Repository (2)



- Under Src:
 - Build: EiffelBuild source code
 - C: runtime code
 - C_library: libpng, zlib
 - dotnet: .NET specific tools for importing .NET assemblies
 - Eiffel: EiffelStudio source code and runtime
 - examples: examples included in EiffelStudio delivery
 - framework: libraries currently used by EiffelStudio. They are potential candidates for libraries
 - help: source code of wizards for project creation
 - Library:
 - tools: various tool useful for developing

How to start digging into EiffelStudio



- Get familiar with EiffelStudio
- Start with either `EB_TOOL` and `EB_DEVELOPMENT_WINDOW`
- Usually names are meaningful therefore doing a regular expression search on class names should yield a positive results
- Web resources:
 - Wiki: <http://eiffelsoftware.origo.ethz.ch>
 - Mailing list: <mailto:es-devel@origo.ethz.ch>
 - Your ETH assistant

Documentation



- Source code for building doc_builder is at trunk/Src/tools/doc_builder
- Documentation is written in XML and then converted to HTML using doc_builder
- For more details read:
<http://eiffelsoftware.origo.ethz.ch/index.php/Documentation>

Repository



- Under trunk you have:
 - Delivery: Files and scripts to build a complete installation of EiffelStudio
 - Documentation: XML representation converted in HTML to produce the EiffelStudio documentation at <http://docs.eiffel.com>.
 - Src: Source code for libraries, frameworks, samples and tools
 - eweasel: regression test tool used for the Eiffel compiler

Runtime



- Handles:
 - Memory management and garbage collection
 - Equality and copy
 - Generic conformance
 - Object traversal
 - Debugging facilities for EiffelStudio
 - Threading

Runtime binaries



- Runtime: C/run-time/lib[mt][ebench|wkbench|finalized].[a|so]
- Ecdbgd: C/ipc/daemon/ecdbgd
- Helper for incremental objects storing in compiler: C/compiler/lib[mt][w]compiler.a
- Helper for debugging: C/ipc/ewb/lib[mt][w]ewb.a
- Helper for launching C compilation: C/platform/libplatform.a

© 2007 Eiffel Software

37

Potential good projects



- Code completion:
 - Add stub routines for inherited deferred routines
 - Add preconditions to a routine by analyzing preconditions of routines used
 - Add predefined code snippet
- Add new type of refactoring
- New wizards to create classes (e.g. if it is a Vision2 window, then add vision2 library automatically to project configuration)

© 2007 Eiffel Software

40

Contributions



- Best contributions will be integrated to EiffelStudio
- What are "best" contributions?
 - Useful for all/most Eiffel programmers
 - Working
 - Clean
 - Documented
 - Elegant design
 - Contracted

© 2007 Eiffel Software

38

More potential good projects!



- Tooltip in editor for both showing routines contract and attribute/local/argument value when debugging
- Redo error and warning reporting
- Detect syntax and semantics errors while typing
- Auto-correction facilities
- Integrate EiffelBuild into EiffelStudio

© 2007 Eiffel Software

41

Already in 6.0



- Tabbed editor
- Fully customizable layout
- Contextual menus instead of pick and drop

© 2007 Eiffel Software

39

More!



- See Wiki:
<http://eiffelsoftware.origo.ethz.ch/index.php/Category:Projects>

© 2007 Eiffel Software

42

Useful links



- <http://www.eiffel.com>
- <http://docs.eiffel.com>
- <http://eiffelsoftware.origo.ethz.ch>
- <https://eiffelsoftware.origo.ethz.ch/svn/es>

Q&A



Any questions?

Thanks and happy Eiffeling!