Architecture of EiffelStudio

Emmanuel Stapf
Eiffel Software
April 17th 2007
Overview of EiffelStudio

- EiffelStudio by numbers
- General overview
- Compiler
  - Validation
  - Code generation
- User interface – EiffelStudio
- Repository
- Runtime
- Projects
- Q&A
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
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
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.
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.
Overview 1 - EiffelStudio’s architecture (3)

Libraries
- EiffelBase
- EiffelNet
- Editor
- Docking
- Gobo (XML, Lex, Parse)
- EiffelVision2

Framework
- Configuration
- Eiffel Parser
- ..NET generation
- ..NET debugger
- Vision2 extensions

Application
- Errors
- Display
- Browsing
- Metrics
- Compiler Core
- Wizards
- Tools
- Search
- Dialogs

UI
Overview 2 - Compilation process

- Degree 6: finding classes
- Degree 5: parsing classes
- Degree 4: inheritance analysis
- Degree 3: type checking
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
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).
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).
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
Other patterns in use in EiffelStudio:
- Observer
- Singleton
- Lazy initialization
- Flyweight
Compiler – AST

- All classes representing AST nodes are descendants of AST_EIFFEL and have the _AS suffix.
- Parser written using gelex/geyacc.
- 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).
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.
All types appearing in an AST are transformed into instances of \textit{TYPE\_A}.

\textit{TYPE\_A} descendants:

- \textit{CL\_TYPE\_A}
- \textit{GEN\_TYPE\_A}
- \textit{TUPLE\_TYPE\_A}
- \textit{LIKE\_FEATURE}
- \textit{FORMAL\_A}
- ...
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`
  - …
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.
Compiler – IDs

A
class_id = 1
routine_id_set = {1}
feature_id = 1
body_id = 1

B
class_id = 2
routine_id_set = {2}
feature_id = 4
body_id = 2

C
class_id = 3

f

++
routine_id_set = {1, 2}
feature_id = 8
body_id = 3
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
**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:
  \[
  a.f \ (\text{args}) \Leftrightarrow \text{routine} \ [\text{dynamic_type} \ (a)] \ (\text{args})
  \]
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
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...
EiffelStudio – Diagram tool

- Uses graph library as data structure for internal representation:
  - Inherits from \textit{EG\_NODE}
  - Supports “physics” (force directed layout)
- Drawing done using model cluster of EiffelVision2 (\textit{EV\_MODEL\_WORLD})
- Two models are supported:
  - BON (\textit{BON\_CLASS\_DIAGRAM})
  - UML (UML subset, \textit{UML\_CLASS\_DIAGRAM})
EiffelStudio - Queries

- Unification of classes/features/metrics facilities through a query language
- Grammar not fully specified yet
- What we have in mind: something like
  
  ```sql
  select classes
  from cluster=base
  where count(features) > 10
  ```

- Work still in progress
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*
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`)
STONE descendants:

- CLASSI石家: non-compiled class
- CLASSC石家: compiled class
- CLUSTER石家: cluster/group/library/assembly
- FEATURE石家: feature in context of a class
- ERROR石家: compilation error
- OBJECT石家: object in debugger

...
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
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
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](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`
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](http://eiffelsoftware.origo.ethz.ch)
  - Mailing list: [mailto:es-devel@origo.ethz.ch](mailto:es-devel@origo.ethz.ch)
  - Your ETH assistant
Repository

- Under trunk you have:
  - Delivery: Files and scripts to build a complete installation of EiffelStudio
  - Src: Source code for libraries, frameworks, samples and tools
  - eweasel: regression test tool used for the Eiffel compiler
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
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
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/deamon/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
Contributions

- Best contributions will be integrated to EiffelStudio
- What are “best” contributions?
  - Useful for all/most Eiffel programmers
  - Working
  - Clean
  - Documented
  - Elegant design
  - Contracted
Already in 6.0

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

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