Introduction to Eiffel programming language

Angel Garcia - aga05001@student.mdh.se
Matti Simperi - matti.simperi@swp.fi
Department of Computer Science and Engineering.
Malardalen University, Vasteras, Sweden

May 24, 2006
Contents

1 Understanding Eiffel 1
  1.1 Implementing ‘Hello world’ ............... 1
  1.2 Features of classes ...................... 2
  1.3 Making Objects .......................... 3
  1.4 Genericity & Inheritance ................. 4
  1.5 Polymorphism & Dynamic Binding .......... 5

2 Differences with other Object-oriented Languages, Benefits & Drawbacks 7
  2.1 How Object-oriented ...................... 7
  2.2 Compiling process ....................... 8
  2.3 Running speed ........................... 8
  2.4 Readability and clarity of code .......... 8
  2.5 Automatic Memory Management .......... 9
  2.6 Over-loading ............................ 10
  2.7 Automatic initializing of member variables 10
  2.8 Pre- and post-conditions and invariants 10
  2.9 Typing .................................. 11
  2.10 Generic classes (templates) .......... 11
  2.11 Inheritance ............................ 12
  2.12 Scope of class features ............... 12
  2.13 Regular Expressions .................... 12

3 More about Eiffel 14
  3.1 When and Why Eiffel was Founded ....... 14
  3.2 What Eiffel is Capable for? ............ 15
  3.3 Real-life Products Made by Eiffel ...... 15

4 Eiffel in the future 17

5 Conclusion 18
Abstract

The introduction to Eiffel language was our topic in Object-oriented Programming course at Malardalen University. The course is taught by Martin Skogevall. The course target is to give to students more sophisticated knowledge about object-oriented programming and introduce some features that are widely used in object-oriented programming projects. These kind of features are e.g. all the design patterns.

The report part of the course is very important. It is way to give students possibility to research deeply in selected topic. Our group topic was Eiffel language. The members of our group didn’t have any previous knowledge from Eiffel, which made this research to be a challenge for us. Our topic had a slightly different approach if you compare it to other topics in this course. The reason was that this report has to be more introductive to world of Eiffel than a specific research of smaller topic.

The material which was used in our report consists of couple of books and few websites. The big problem with printed material, which we were able to use, was, that is was all quite old. Some of them were even published in 80s.

Eiffel seems to be minority language in whole software engineering industry, but there are some special features, like built-in design by contract methods, which is the reason why some specialized companies are using Eiffel to produce high quality software projects. Even during this short introduction that we had to Eiffel language made us to think its possibilities to design and develop high quality software.

The quick experience of Eiffel was so positive that we wouldn’t remove possibility to even do some small project with Eiffel in feature. The biggest problem against trying Eiffel would be that it is so minority language and doesn’t have enough reputation as good language in today’s business.
Chapter 1

Understanding Eiffel

Eiffel language is based in the Object Oriented Programming concept and it is designed with the explicit intent to produce high quality, reliable software. Its structure is based in Classes (compile-time notion) and Objects (only exists at run time). The purpose of this paper is not to be an introduction to Object Oriented Programming (O.O.P.) paradigm but how Eiffel language uses it. For that reason, be familiarized with O.O.P paradigm can be helpful to understand better the next concepts.

1.1 Implementing ‘Hello world’

To get familiarized with Eiffel environment we can write the typical ‘Hello world’ program using EiffelStudio 5 (5.6.1218 Free Edition). Then, looking through the code we can differentiate the main parts in an Eiffel program. A good example can save thousand of words.
1.2 Features of classes

Eiffel uses the classes as the main data structure to programming. A class is an implementation of an abstract data type. It describes a set of run-time objects, characterized by the features (operations) applicable to them, and by the formal properties of these features [1].

When a feature is called the common way is to use the dot notation as target.feature_name form but it is also available the possibility of Prefix and Infix forms. There are two kinds of features:

- Routines: represent computations applicable to instances of the class. These routines can be divided into procedures and functions. The procedures perform an action without returning any value and functions make an action returning values.
- Attributes: represent data items associated with the instances of a class.

The next example illustrates how a class with their features can be defined:

```eiffel
class CAR
  feature
    total_speed: INTEGER
    owner: PERSON
    minimum_wheels: INTEGER is 4
    buy (who: PERSON) is
      do
        owner := who
      end
    speed (amount: INTEGER) is
      do
        add(amount)
      end
  feature (NONE)
    add (amount: INTEGER) is
      do
        total_speed := total_speed + amount
      end
end --CAR
```

The class CAR has two features clauses. The first one begins only with the keyword feature without qualification; this means that the features declared inside are available to all clients of the class.

In the second clause the NONE qualification is expressed. What it is written inside the braces is a list of classes that can use that features. The
class NONE it doesn’t have instances so the add routine of that feature it can be only accessed locally by the other routines of the class CAR.

### 1.3 Making Objects

Objects are called the direct instances of the class. In Eiffel’s object oriented programming style any operation is relative to a certain object. If an external client wants to invoke an operation then the name of the object it should specified on the left of the dot.

An object is an instance of a class and in Eiffel is called entity. There are two types of objects: References and Expanded objects. As Reference is the default type declared in Eiffel, we are going to see it closely.

```
y : SOMECLASS;  -- default value is void (entity)
!y.make;       -- call to SOMECLASS creation routine (create the object)
y.routine;     -- routine is a member of SOMECLASS (procedure or function)
```

In the first line the value for the entity y is void. By default, entities are void initialization. This is similar to NIL in Pascal or NULL in C/C++ languages.

![Object Initialization with void](image)

The double exclamation operator (!!) is used to obtain the object at runtime. The object is instantiated and it is initialized according to the make routine of SOMECLASS. Then, the object looks like:

![Object Instantiation](image)

If a class is defined so that all instances of it are expanded then, the objects can be used as soon as they are declared. For example, when an object of REAL class is declared, it is initialized automatically (without using !! operator as Reference type) because REAL is a expanded type class.

To declare a class in which all their instance are expanded we use the keyword ”expanded” in the before the class declaration. See example:
Now, to instance and object of the class we write:

```plaintext
y : INTEGER;    -- default value is 0 (object)
y := 1;         -- y now contains the value 1. (object)
```

### 1.4 Genericity & Inheritance

Eiffel uses the concept of classes to build systems with a solid architecture but in order to ensure the reusability and extensibility characteristics, it uses two fundamental techniques: Genericity and Inheritance.

To kinds of parameters must be specified to implement the Generacity mechanism.

- **Formal generic parameters**: represent arbitrary types to the class that is defining. In the example, the formal generic parameter \(G\) represents this type.
- **Actual generic parameters**: specifies a directly usable type. It must be provided a concrete type corresponding to \(G\).

In the next example we can see the LIST class with a generic parameter \(G\).

```plaintext
class LIST [G]
    feature --Access
        item: G
    end

    feature --Element change
        put (new_item: G)    -- Add 'new_item' at the end of the list
    end

end --LIST
```

Now, we redefine the general parameter to a concrete type when instance the class:

```plaintext
my_cars: LIST [CAR]                    -- A list of my cars
saab, volvo: CAR
biketam: BIKE

my_cars.put (saab)
my_cars.put (volvo)
my_cars.put (biketam)  -- error
```
The inheritance mechanism allows defining a new class by combination and specialization of existing classes. Eiffel supports multiple inheritances. A class inherits features from one or more parent classes, declared in the inherit clause of a class definition:

```eiffel
class CHILD
  inherit
    PARENT_1;
    PARENT_2;
  feature
    ... -- own features of CHILD
end --CHILD
```

### 1.5 Polymorphism & Dynamic Binding

In the imperative languages, an assignment like `a:=b` can be done only if the two variables belong to the same type. Sometimes a conversion “cast” is needed when `a` and `b` have different types.

Using Polymorphism in Eiffel, `a:=b` is permitted not only if `a` and `b` are of the same type, but more generally if `a` and `b` are of reference types `A` and `B` based on classes `A` and `B` such that `B` is a descendant of `A` [1, p16]. Then we can conclude that a more specialized type may be assigned to an entity of a less specialized type but not in the opposite way. In the next example the class `CAR` has a `total_weight` routine (in this case is a function) into feature section. Then, we can redefine it in the subclasses inherited from class `CAR`.

```eiffel
Class MINICAR inherit CAR
  redefine total_weight end

  feature
    num_passengers: INTEGER is 1
    weight_passengers: INTEGER
    total_weight: REAL is
      -- a kind of MINICAR
      do
        Result:= num_passengers * weight_passenger
      end;
    .. other MINICAR feature ..
end MINICAR
```

Another kind of “minicars” can be used redefining also the `total_weight` function, then, the version of ”minicar” to use it will determinate at run-time mode.

Polymorphism in Eiffel is safe because is restricted by inheritance, this means by the hierarchical structuring of classes (we can only assign from a descendant to an ancestor type); see next figure:
The assignment c:=mc is valid because MINICAR class inherits from CAR class. In run-time mode, the boolean variable cond in the conditional sentence can take the values true or false. When it takes "false" then c is attached to CAR type and it will use the CAR definition of the total_weight routine. But when cond takes the "true" value then c is attached to a MINICAR and now it will use the MINICAR definition to total_weight routine. This behaviour is known as dynamic binding. Note that the choice can only carry out in execution mode.
Chapter 2

Differences with other Object-oriented Languages, Benefits & Drawbacks

This chapter will cover some distinctions between Eiffel and other object-oriented programming languages. Comparisons are made e.g. in some general things like syntax and speed and in some object-oriented features that are expected to be implemented someway in every object-oriented language. Because of the nature of programming language, even some basic features might have totally different way to be implemented if they are even implemented at all. Two major programming languages where comparisons are made are Java and C++. Both of them are used very widely in software industry around the world and are though even as ”standard languages”.

2.1 How Object-oriented

Eiffel, C++ and Java are all object-oriented languages, but that is not enough. Everybody knows that C++ has quite few features that broke purity of object-oriented languages. Nobody can deny those, but with Java it is different. Java is commonly known as pure object-oriented language. This is defined because everything is happening inside objects. Every program need to have at least one object that system is creating and using to start the program.

Eiffel Software sees the situation little bit different. Even if everything is happening inside objects, that is not enough to specify language as pure object-oriented language. Actually Eiffel Software comparisons report is saying that pure object-oriented language shouldn’t have any construct similarities with other non-pure object-oriented language like C. That is the reason that they are looking Java more like on C extension. [5].
It is important also notice that all basic variable types in Eiffel are already Objects. There is no way to create other variables then objects. E.g. Java includes some primitive types like integer, double and char which are actually objects.

2.2 Compiling process

Java programs are always compiled first to Java byte code. After that is can be executed on-the-fly trough Java Virtual Machine. This is fairly fast in development process, because it is possible to just compile very small part of the code. With C++ the program is usually compiled without any intermediate form to operating system’s code as a new executable. [5] There are often some files which can be used in compiling process without starting from ASCII code, e.g. DLL files, but however the executable files are created again.

Eiffel is using combination of interpretation and compiling. [5] It is possible to convert Eiffel code to intermediate form which can be converted to Java byte code or even optimized with C compiler.

2.3 Running speed

All the common programming languages can be compiled with reasonable speed to ready to run programs. It doesn’t make any different with small differences, because this is made only once and then the program can be executed many times. The differences can be founded on speed to execute code after compiling.

C++ is known as a quite fast code and Java on the other hand little bit slower. So where does Eiffel locate in this comparison. The safest way to say is that Eiffel fits in the middle. Eiffel is not as fast as C++, because of automatic garbage collector, but it is faster then Java. [6] The loading speed of Java is very poor, because it is only interpret by the "compiler" to Java byte-code, not as a language that operating system is understanding directly without any specific running environment (e.g. Java Virtual Machine).[5]

2.4 Readability and clarity of code

The basic needs for syntax are readability and clarity for the programmer. The compiler needs to be sure what programmer wants different part of the code to do. That is the only reason why we need to have a specific and
actually quite simple syntax (compare to human language). C++ and Java have both syntaxes, which are very similar to C programming language. C++ syntax is even little bit closer, then Java’s syntax, to C language’s syntax.

Eiffel Software’s comparison [5] is stating that Eiffel language is much easier to read for the programmer. This seems to be little bit sharpened in a start, but most of the people who are comparing these languages are looking them with eyes of programmer, not like a common people who doesn’t have any skill in computer science. For programmers it is not hard to read different languages if the behaviour is similar. You are expecting something and you will easily recognize different elements in code.

Think about the whole situation with different way. Big software project includes many peoples with different area of expertise, but everybody needs to understand each other. Mainly this can be done fairly easy with different information system modelling languages, e.g. UML (Unified Modelling Language). But when there is need for specific algorithms or mathematical formulas, we run into problems. If you need to comment from financial analyst or expert of other area, it is very important that syntax is easy to teach and start understanding in short period of time. In this situation Eiffel is easiest to understand comparing to C++ or Java. [5]

2.5 Automatic Memory Management

Automatic memory management means that programmer doesn’t need to pay attention for destroying variables/objects after there is no use for them anymore. If environment has implemented automatic garbage collector, system will automatically destroy those memory areas which are not needed by the program anymore. This is done by removing all the references to those memory areas. If language doesn’t implement automatic garbage collector, programmer’s needs to pay a lot attention for keeping book for used variables/object and remember to delete them after they are not used anymore. With both ways, after freeing memory, same memory can be used by other users.

C++ doesn’t have any kind of automatic garbage collector in commonly used programming environments. Java uses automatic garbage collector. Eiffel uses also automatic garbage collector. In multi-threading environment Eiffel has own garbage collector for every thread, which is not usual. This basically means that every thread can reserve memory when they need
without interrupting other threads. This kind of solutions is important e.g. in real-time systems. [5]

2.6 Over-loading

Over-loading means that basically that we can define in our program several meaning for an indication. The right meaning is selected after we know which information is used with this indication. An indication could be an assign operation or a method call. In function/method call, the system could decide which of the methods should be used according to arguments’ types that were given in this call.

C++ uses widely over-loading. It is possible to over-load both operators and functions. Over-loading of operators is done with specific syntax that is defined similar to function. Java does allow only over-loading of methods. There might be several methods in one class which have all same names, but the whole signature of method is different because of arguments. The variation can be either the types of arguments or the number of arguments or both of them. An in the end, Eiffel doesn’t allow over-loading of methods or operators but it allows creating new operators. [8] It could be possible to just rename methods accordingly to their arguments type like useOfTheFunction_int like it is handled in some scripting languages sometimes. This isn’t the most sophisticated way as it is easy to notice.

2.7 Automatic initializing of member variables

Eiffel has special kind of way to automatically initialize member variables with default values, when they are created. E.g. integer will be initialized with 0 and Boolean with false. This is somewhat safer way for the programmer, because then there won’t be so much unexpected behaviour with program in development process. [9, p. 76-77]

In C++ and Java programming languages this can be done inside constructors or variables’ definition part. There won’t be any automatic initializing so it is up to the programmer to take care of it. Forgetting to initialize variables is very common mistake and sometimes it is pretty difficulty to notice it and correct it, because the behaviour can’t be seen easily.

2.8 Pre- and post-conditions and invariants

Eiffel includes very efficient way to debug programs and detect problems in methods by defining pre- and post-conditions. These can be used to detect different variable values that they are correct when entering to method or
when leaving method.

Requirements inside a method can be defined with require elements. If the requirement is not filled, the system will give an error message. [10, p. 4-5] The make a requirement for return value, programmer has to use ensure element, which is checked against return value [10, p. 5-6].

Invariants means that the requirements that are specified as invariants has to be filled every time after some public method was called. These invariants are described once for the class. The element name is invariant in Eiffel languages. Invariants are presented because otherwise it would be difficulty to specify all the requirements to every method and it would be hard to change all the requirements in every method if there is a small change in code. [10, p. 6-7]

All of these requirement specifications are very useful, because then there is standard way to check the state of variable or object. Objects and variables can be checked now against the rules specified in documentation which are created in design process of information system. Then a notable amount of errors will be noticed little bit easier without defining lot of more if-else-conditions around to code.

2.9 Typing

Language can support either strong or weak typing. Strong typing means that all the types are predefined and compiler will have knowledge of them. There isn’t way to find out the type of object or change the type of object during runtime. Strong typing makes code more consistency. With weak typing it is possible to change the type of variable during runtime, i.e. casting.

Eiffel uses strong typing, which has comes up with some limitations, but means robustness in compiling. Both C++ and Java are using weak typing, because there is way to find out the type object and change the type of variable in runtime with casting. [11, p. 10]

2.10 Generic classes (templates)

Generic classes mean that there is a possibility to define a template where programmer doesn’t specify some variable’s type. This type is derived by use of this template. This gives a fairly easy way to describe same kind of algorithm to different types.
Eiffel language gives a way to define templates as does C++ languages. Java language doesn’t have this possibility. [8]

2.11 Inheritance

All object-oriented languages have some way to inherit super class’s features. The can be possibility to inherit one or more classes by one class. Those are called single or multiple inheritances. If one class has a possibility to inherit many classes, can the structure of the program become much more complicated, then with only one inherited class. Multiple inheritances may also arise a lot of problems with names, because both of the inherited classes can have variables or method with same names.

C++ and Eiffel support multiple class inheritance, where Java supports only single class inheritance. Java supports also multiple interface inheritance. [11, p. 17] Eiffel has very efficient way to handle problems, which could arise when having same names with features. There is specific rename clause which gives a possibility to the user to handle these problems.

2.12 Scope of class features

Scope of class features means that who is able to see and change class features. Commonly these are divided with three different clauses, private, protected and public. Private means that only same class’s methods can access to this feature. These kinds of variables can’t be over-written in class which will inherit this class. The different between protected and private is that protected methods can be over-written by a class which inherited current class with protected method. Public variables and methods can be accessed everywhere and methods can be over-written.

In Java and C++ it is possible to define class features with all of these different scope areas, but Eiffel only supports protected and public. But with Eiffel there is way to specify which of the features programmer wants to inherit. It is possible to redefine also some features e.g. make a normal integer as a final integer in subclass. [12]

2.13 Regular Expressions

"Regular expressions are pattern matching constructs capable of recognizing the class of languages known as regular languages. They are frequently used for text processing systems as well as for general applications that must use pattern recognition for other purposes.” [8]
Regular Expressions are widely used and needed in many different situations. Some of the languages have native support and some have implemented it in standard library. Java includes support for regular expressions in a standard library. Eiffel and C++ have a support in libraries, but which are not standard. [8]
Chapter 3

More about Eiffel

As it was commented before, Eiffel is a programming language develop to achieve the best results when quality software is treated. Due to its nature, in Eiffel every entity is an object declared of an explicitly stated class type. The purpose of Eiffel is to help specify, design, implement and change quality software. This purpose can be achieved by a combination of many factors:

- Reusability is the ability to produce components that may be used in many different applications.
- Extensibility determines how the software is supposed to be, it is very hard to modify software systems, especially large ones.
- Reliability: Eiffel’s approach to the engineering of quality software includes some techniques such as assertions, disciplined exceptions handling and static typing that enable developers to produce software with few bugs.

Satisfying the two previous factors means having less software to write and then spend more time to focus in other goals such as efficiency, ease of use or integrity. In order to achieve Reusability, Extendibility and Reliability, the principles of object oriented design provide the best known technical answer.

3.1 When and Why Eiffel was Founded

TEiffel was created in 1985 by Bertrand Meyer. He is the President of Interactive Software Engineering in Santa Barbara (California) and Société des Outils du Logiciel (Paris). He is also Chairman of the TOOLS (Technology of Object-Oriented Languages and Systems) conference series.

Eiffel is an Object Oriented Programming language focused on developing robust and quality software. Its syntax remembers us to Pascal or
ALGOL language, it has a strongly statically typed and it uses the automatic memory management (well-known as garbage collection).

3.2 What Eiffel is Capable for?

Eiffel is capable of everything. There are no specific limitations that Eiffel is not capable. Eiffel is pure system level programming language which can be used as language like Java and C++ to develop all kind of programs. Eiffel has even some remarkable features that promote its advantages as a programming language. The main focus with Eiffel has been always to produce fast, but also high quality, software. There are hundred of thousand of software projects where Eiffel programming language has been used satisfactorily.

3.3 Real-life Products Made by Eiffel

OHere we will present couple of projects that Eiffel Software is presenting as on example cases in their website at www.eiffel.com. The cases are "Tech Success: Xontech uses Eiffel for defense", "Fortune 500 Technology Manufacturer Uses Eiffel Software to Dominate its Market" and "Small Application Development Company uses Eiffel to Dominate Its Market". These cases are written by the Eiffel Software company and that’s way they can’t be though as an objective opinions.

First case study concerns Xontech. The project group of 20 persons finished in 5 months a ballistic missile simulator, which they said would have taken much more with other language and programming environment. The reason why they selected Eiffel, which is not the major language today in software engineering industry, was that they wanted to design and deliver quality software quickly as Eiffel was promising. "It is a very transparent language. I’m never struggling with the language, just with the inherent problem I’m trying to solve", said Darren Hiebert as a characteristic for Eiffel programming language. They used EiffelStudio as their development environment.

Second case study is about EMC, which is one of the leading companies in storage management line of business. Their need was to release new critical product group companies as a target group. By selecting Eiffel they say that they gained lot of benefits where the most important were that is solved their engineering problems and increased productivity and made their systems very reliable. The product family’s revenue has increased up to $1 billion in 7 years of start. The project team was forced to use some components which were made with C or C++ to be able to use e.g. SSL-
encryption. This wasn’t a problem and they could integrate them easily with rest of the project. The results for the project were quite similar to the first case. The gained some increased speed in development and they could use smaller development teams.

The third study case is about small-size company Fowler Software Design LLC and they have made successful growth in software business area. Fowler is an independent company which focus is to create to small and middle size companies’ software like CRM or financial management. The founder of Fowler, Rex Fowler, stated that they know very well different programming languages and they are very pleased with Eiffel’s reliability, reusability and high quality. These were and are the reason why they are using Eiffel language nowadays and a main programming language in their company. Fowler Software has created their own framework which they are using widely in their current and future projects. This reusability of code helps them to decrease time and money spent in development process.
Chapter 4

Eiffel in the future

Until recently, only one company (Interactive Software Engineering) was the sole supplier of Eiffel compilers, and also one person (Bertrand Meyer) was in charge of the language definition. Now the situation it is totally opposed: the Nonprofits International Consortium for Eiffel (NICE, we think they are nice too) has been created to manage the new challengers as competing implementations, control over the future of the language and also an organization representing the users and implementers of Eiffel worldwide. The future of Eiffel is open and everyday is growing with new possibilities and new functions showing that Eiffel can be a relevant chance to develop quality software. After read this paper, the reader can participate with more ideas and coding to this exciting endeavour.
Chapter 5

Conclusion

The Eiffel Language is quickly emerging as the language of choice for developers of quality software because of it combines the rigorous software engineering principles with advanced object-oriented programming techniques. It is used in a wide range of application areas and its attracted considerable attention is not just as an implementation language but also as a high-level notation for analysis and design [1].

There is no doubt that Eiffel programming language provides a good chance to develop robust and feasible software in a wide application field. As P. Stephan from CALFP Bank says, the readability of a programming language is very important. Someone who has expertise in a business domain understands its underlying concepts; In the same manner, these people can express business concepts easily in Eiffel because they can focus on design and implementation, rather than struggling with memory management problems and debugging such as C and C++ programmers do. Eiffel is fully backed by strong software engineering concepts: reusability, extendibility, and reliability. It allows you to evolve your projects rapidly, without fear...If it compiles and runs properly with assertions on, there is no risk that you have lost or overlooked something. It reusable structure permits to software developers to increment faster the production of software components rather than one-of-kind applications.
References

http://es.wikipedia.org


[5] "Object-Oriented Languages in the Industry A Comparison” - Eiffel Software company

[6] "comp.lang.eiffel Frequently Asked Questions (FAQ)” - (Many authors)
http://www.faqs.org/faqs/eiffel-faq


[8] "Programming Language Comparison” - Jason Voegele
http://www.jvoegele.com/software/langcomp.html

