# Concurrency in Eiffel
# Topic Paper #19 (Term Paper)

Alex Laird
CS-3210-01
4/20/09
Survey of Programming Languages
Spring 2009
Computer Science, (319) 360-8771
alexdlaird@cedarville.edu

| Grading Rubric | | Max | Earn |
|---|---|---|---|
| | On Time/Format | 1 | |
| | Correct | 5 | |
| | Clear | 2 | |
| | Concise | 2 | |
| | **Total** | 10 pts | |

| Peer Reviewer |
|---|
| |

## ABSTRACT
This paper discusses in depth the elements of concurrency, the Eiffel programming language, and specifically concurrency in Eiffel.

## Keywords
Concurrency, Eiffel, Programming, Language

## 1. INTRODUCTION
Concurrency in the realm of computer programming is the concept of completing multiple processes at the same time (concurrently). Eiffel, developed in 1986 by Bertrand Meyer, is a statically typed, object-oriented programming language that is known for having strong support for concurrency [4].

## 2. FOUNDATIONS OF CONCURRENCY
When programming is concurrent, multiple computations are executing simultaneously, potentially interacting with each other. This may happen on multiple cores on the same chip, time-shared threads on the same processor, or on physically separated processors, even across a network. At the highest level, these multiple processes running concurrently are what concurrency is. Generally, each process is referred to as a thread [2]. Another term for concurrency could be parallel processing, though the term concurrency envelopes more than just parallel processing.

Since the entirety of a computer's processing power is almost never fully used, especially with the large-scale processors in production today, a vast amount of processing speed is wasted when any application is run. When the application is launched, it opens an initial thread that allows it to use a certain number of resources on a specified processor. A well-developed program that makes use of threads will launch additional threads throughout the program to complete tasks in the background while the user continues using the application that is running on the main thread. Concurrency allows a program to continue running, send a computation out to another process that the user never knows about, and then return the result to the user in a timely fashion. The beauty of it is, if implemented efficiently, the user may not even realize when massive computations are done. The principles of concurrency are synchronous processing and shared resources [4].


*Alex Laird, Cedarville University, Cedarville, Ohio, 45314*


## 3. PROCESSES IN CONCURRENCY
When a program is running concurrently, all of its processes must operate, interact, and synchronize with each other in a proper fashion. Some languages allow this more easily than others, but proper communication between two processes allows one process to influence the execution of another. If two processes do not communicate properly, this may cause a failure to execute in another process, or it may cause another process to execute improperly, thus rendering the wrong result or a runtime error [2].

Processes occur at unpredicted speeds and times. There is no guarantee that two processes will execute before or after each other. This being the case, processes must be synchronized and communicate properly if they are to produce the desired result. To do this, two processes that need to communicate should use shared variables. The shared variables keep track of the state of each process so another process does not attempt to manipulate anything before it is allowed. Essentially, synchronization in concurrency is a set of constraints on the ordering of events. When necessary, the programmer must specify a synchronization mechanism in order to delay certain events until the prerequisite conditions are met [2].

Of course, parallel processes *can* launch in a specified order. The key is that, if order is critical, the preprocess conditions are met before a process is launched.

## 4. RESOURCE LOCKING AND UNLOCKING
Concurrency incorporates the sharing of resources, especially across a network or in a shared database. When these resources are in use, the program using them may lock them if it requires the resource in its entirety without any exceptions. An example of a resource needing to be locked may be two computers trying to manipulate the same file. Concurrency would allow the file to be viewed read-only by multiple systems, but it is locked to editing except by one of the systems, whichever system got to it first [3].

When a system is done using the locked resource, it can release the resource back to the shared database to be used by another system. When it is released, this is called unlocking the resource [3].

## 5. THE DINING PHILOSOPHERS PROBLEM
The Dinning Philosophers Problem has become one of the most prominent examples of concurrency in Computer Science. It was first introduced by Edsger Dijkstra in 1965 as a synchronization problem of five computers competing for access to five shared tape drive peripherals; later it was retold as five philosophers sitting around a table. This problem illustrates deadlock, and

resource starvation assuming each philosopher takes a different fork and each wants to eat [5].

Each philosopher sitting at the table can be doing one of two things: eating or thinking. If they are eating, they are not thinking; if they are thinking, they are not eating. In the center of the circular table is a large bowl of spaghetti, and sitting to the right of each philosopher is a fork. This being the case, there is a fork both to the right and to the left of each philosopher. The philosophers understand that they must have two forks in order to be eating the spaghetti. If each philosopher holds his left fork, none of them will be holding two forks, therefore none of them will be eating. This creates the problem of deadlock.

Since there will not be enough available forks for each of the philosophers to be eating at the same time, this problem illustrates our concept known as concurrency. A philosopher can take two of the forks, thus illustrating a program locking a resource, but then when another philosopher wants to eat he must wait for the eating philosopher to unlock his fork and free the resource so he can use it.

There are many solutions to the problem, but essentially some external source (in this example, for instance, we could say a waiter) must be introduced to inform the philosophers when a resource (fork) is free or not. The waiter would be the only person aware of the status of every fork at the table, and each philosopher would have to ask the waiter for permission to lock a particular second fork for himself [5].

## 6. CONCURRENCY IN SPECIFIC PROGRAMMING LANGUAGES

Concurrency is a standard construct in many languages and an extension to others. Depending on the language and the programming environment, concurrency may be easier or more difficult to achieve.

The most commonly used languages today have at least some sort of construct for concurrency. Java and C#, two of the most widely known interpreted languages in use today, support threads in the default libraries. C++, like many other languages, is one that have extensible support for threads through POSIX, but only if the library is included. For instance, in C++, if you are compiling on a Windows-based system, you will probably have to include the pthreads library. However, if you are developing in a Unix-based environment, pthreads are probably just an include statement away.

Some of the most popular programming languages that have concurrent attributes integrated into the development are Ada, MultiLisp, Io, Concurrent Pascal, and Scala. There are many languages that are not concurrent by nature, but are made concurrent through some alternative implementation. For instance, Concurrent Pascal or Stackless Python.

## 7. THE EIFFEL PROGRAMMING LANGUAGE

Bertrand Meyer developed Eiffel to be an object-oriented, extendable, and efficient programming language that includes a basic class library, integrates easily with other programming languages, and supports such tools as configuration management, documentation, and debugging. Portability was also a large player in the design implementation of Eiffel [6]. Eiffel was influenced by such languages as Ada and Simula, and it has influenced many other languages such as Java, Ruby, and C# [7]. Other key features integrated directly into Eiffel include parallelism, when two processes are executed at the same time, thus running concurrently, garbage collection, and support for multiple inheritance, something that stems from Eiffel's intention to be reusable [6].

The intended audience for Eiffel is partially students interested in learning the fundamental principles of object-oriented programming and developers looking to develop efficient medium to large-scale software projects. Eiffel has generally shunned coding shortcuts and optimization tricks to help the compiler; it emphasizes readability to the programmer, and it also encourages the programmer to focus on the important aspects of a program without getting hung up on syntactical and implementation details. The simplicity of the Eiffel coding style is meant to promote simple, extensible, reusable, and reliable computer programs [7]. The syntactical aspects of Eiffel are most similar to C++, Java, or Python. If someone is familiar with any of these object-oriented languages, learning Eiffel is fairly simple.

There are many compilers available for Eiffel, one GPL (General Public License) distributed compiler is called SmartEiffel and supports many features that are not found in the commercial Eiffel compilers. The largest, most supported commercial compiler is the licensed EiffelStudio, developed by Eiffel Software [10].

## 8. OBJECT-ORIENTATION IN EIFFEL

As previously stated, Eiffel is an object-oriented language. Eiffel is a purely object-oriented language, but it does also provide an open architecture for interfacing with external software developed in other programming languages that may not purely object-oriented or even object-oriented at all. However, wanting to keep up with the current standard, Eiffel was written to be purely object-oriented for a few reasons [6][7].

- The emphasis of Eiffel is around the objects it manipulates, not the functions it performs on them. Specifically, Eiffel emphasizes reusing the objectified structure as a whole rather than the isolated procedures.
- Objects are instances of abstract data types. This way data structures are known by their interface instead of through their representation.
- Classes, as in most object-oriented languages, are the basis of an object representation. The class describes the implementation of the abstract data type.
- Classes are developed as collection units that are interested in a particular topic and useful for manipulating that topic independent from the overarching system.
- Structures illustrate important relationships between classes and objects, particularly the multiple inheritance relation.

The object-oriented nature of Eiffel is important because classes assist greatly in the implementation of threads and concurrency.


*Alex Laird, Cedarville University, Cedarville, Ohio, 45314*

Eiffel enables and encourages the expression of formal properties of a class through assertions. Assertions may appear in three different orders for a class:

- Preconditions. These conditions in the assertion must be satisfied before a routine is called. Preconditions are introduced by the keyword "require."
- Postconditions. These conditions require that particular conditions have been satisfied after the return statement of a routine happens. Postconditions are introduced by the keyword "ensure."
- Invariants. Class invariants must be satisfied by objects of the class at all times, and especially after the object has been created or a routine is called. They help maintain general consistency within an object and are specified in the invariant clause of the class definition.

Assertions assist in maintaining the reliability of the object-oriented Eiffel code.

# 9. EIFFEL COMPARED TO OTHER LANGUAGES

Eiffel programs are compiled like C++, not interpreted like Java, and they directly compile via C to native code. This being the case, Eiffel programs have speeds relatively comparable a C and directly comparable to C++ programs. Obviously, since Eiffel is not interpreted, running speeds are significantly faster than Java applications, both in speed and in memory usage. Like Java, Eiffel incorporates the resource-friendly construct of garbage collection. Unlikely Java, Eiffel's type system is stronger, more robust, and safer.

Eiffel supports generics like Java (known as templates in C++), though they are far more efficient than either Java or C++, since Eiffel has had them built into the construct of the language from the beginning and they are added features to Java and C++ [10].

Eiffel was largely influenced by both Java and C++, so the syntax is strikingly similar to both. However, unlike either of the languages and more like Python, Eiffel does require bracket ({ and }) to open and close sections. You may use brackets, but the defining mark of a section is an indentation, and a section is closed by the keyword "end." Additionally, though there are broad rules for how case must be handled, Eiffel is not a case sensitive language. In general, Eiffel prides itself on having a considerably more readable and pretty syntax than Java, C++, or even Python.

Like all of the most popular languages, Eiffel does support exception handling. However, Eiffel sees catching and handling exceptions gracefully a crucial element in elegant programming, and since Eiffel strives to be a simplistic language, and especially tries to be a good language for teaching beginners how to program, Eiffel takes exception handling one step further than most languages. It incorporates a "rescue/retry" strategy for error recovery.


*Alex Laird, Cedarville University, Cedarville, Ohio, 45314*


# 10. CONCURRENCY IN EIFFEL

Eiffel is known as a language that easily incorporates concurrency. The language itself builds particular processes around each other into subsystems. If necessary, one object may call two different subsystems and they will be executed as separate threads at the same time. Since one of the main goals specified in the development of Eiffel is parallelism, and parallelism is a direct descendent of concurrency, Eiffel is known as a concurrent language by default. However, since Eiffel programs are compiled on concurrent levels and no large-scale support is implemented directly into the language, it can be slightly more complicated to make your own programs concurrent beyond what Eiffel does automatically at compile time [9].

When Eiffel compiles two things that it wants to run parallel to each other, it makes each one of them a PROCESS object and creates a new thread execution for it. The limited PROCESS object allows some simple manipulation, the most significant of what is the *wait-by-necessity* principle. The *wait-by-necessity* command is invoked only when a new process attempts to use the result of another process that has not been completed yet. Beyond this simple rule, other concurrency packages must be used. Every commercially developed Eiffel IDE provides some mechanism for concurrency, but the most widely used concurrency extension for Eiffel is the third-party developed SCOOP.

# 11. SCOOP AND EIFFEL

Currently, fully-featured concurrency is most easily achieved with the use of the additional packages provided by SCOOP (Simple Concurrent Object-Oriented Programming), but EVE (Eiffel Verification Environment), the Eiffel research group for future developments in Eiffel, is striving to integrate the SCOOP concurrency packages into future releases of the Eiffel programming language [8][10]. SCOOP is the easiest way to use threads in Eiffel.

SCOOP was designed specifically for Eiffel, but it's designed in such a way that any programming language fitting the specifications for it would easily use it [8].

Attaining concurrency deliberately for an Eiffel object is far simpler than any of the languages predecessors. Simply appending the keyword "separate" to the beginning of a class definition. When the compiler sees this, it will recognize this as a parallel class and give it the instructions to be run concurrently when called in runtime. Using SCOOP, things called processors are used to explain the execution of a separate object, and the "separate" keyword is used to instantiate the execution [9].

SCOOP allows for the reserving of objects and resources, similar to locking and unlocking. For instance, if two threads that are to be executed at the same time are going to manipulate some third thread, and the other in which threads one and two are executed depends on the result manipulated in thread three, the third thread's resource must be reserved locked by thread one. When thread two sees that the object is reserved, it will simply *wait-by-necessity* until the object becomes unreserved. Then and only then will it perform its operation on the object and continue on to thread three [9].

SCOOP is simple because it introduces only a few key concepts on top of the object-oriented model. SCOOP is object-oriented because it maintains and expands upon the already present object-oriented principles withheld in Eiffel or a similarly appropriate

programming language. SCOOP is simple and concurrent because it allows for the easy use of concurrency, threads, and parallelism without forcing the programmer to meddle in the painful errors and headaches that come with attempting to properly annotate and synchronize a program before it can run concurrently [8].

## 12. PRAISE FOR EIFFEL

As stated previously, Eiffel does have a compiler under the GPL, which means it is open-source and cross platform. The standard IDE, EiffelStudio, is available for numerous platforms, and there are dozens of other IDEs and compilers that will write and compile Eiffel code into binaries or bytecodes for any platform.

Much like Java, Eiffel has very well documented in APIs (Application Programming Interfaces) online how the features of the included libraries work and can be used. Additionally, there are numerous tutorials provided online for free at Eiffel's official website. Also similar to Java's Javadoc, Eiffel has the ability to extract documentation automatically. This is an extremely important feature, especially in large-scale development and when working on with a team [10].

Eiffel helps you catch your mistakes. With the more recent addition of AutoTest, Eiffel has become increasingly better at catching its own mistakes as well as yours. Whether the mistakes are syntactical or runtime generated, AutoTest will do its best to catch them. Even if the problem isn't any sort of an error, perhaps it simply isn't optimal for performance, AutoTest may catch it. In fact, after implementing AutoTest for the first time, Eiffel Software found many bugs in its standard libraries that it had been using for years [10].

EiffelStudio comes with EiffelVision, a GUI (Graphical User Interface) developing utility that allows the programmer to easily and efficiently create a Windows, GTK+, or Mac OS X visual interface. Additionally, EiffelStudio doesn't just allow compilation of binaries to numerous platforms via C and C++ libraries, it also allows for the compilation of bytecode to Java or most things on the .NET framework, including C#.

## 13. CONCLUSIONS

Eiffel is not a new programming language by any means, though it is younger than its grandparents, Java and C++. Even still, it's a wonder the language has not seen more mainstream attention with its simplicity, feature richness, and attention to documentation and detail. Perhaps if the designers at Eiffel Software continue developing Eiffel into an even more powerful language with cross platform binary compilation support, emulation bytecode compilation, multiple inheritance that doesn't cause endless headaches, neatness and auto-correction, and almost seamless integration with concurrency, Eiffel will be able to give Java a bit of a competition in the near future. Until then, Eiffel will remain one of the best-kept secrets of the elite programmers around the world.


*Alex Laird, Cedarville University, Cedarville, Ohio, 45314*

## 14. REFERENCES


[1] Sebesta, Robert W., 2008. Concepts of Programming Languages. Addison-Wesley, Boston. ISBN: 978-0-321-49362-0

[2] Andrews, G. R. and Schneider, F. B. 1983. Concepts and Notations for Concurrent Programming. *ACM Comput. Surv.* 15, 1 (Mar. 1983), 3-43. DOI= http://doi.acm.org/10.1145/356901.356903

[3] Thomasian, A. 1998. Concurrency control: methods, performance, and analysis. *ACM Comput. Surv.* 30, 1 (Mar. 1998), 70-119. DOI= http://doi.acm.org/10.1145/274440.274443

[4] "Concurrent Computing." Wikipedia, the free encyclopedia. 20 Apr. 2009 <http://en.wikipedia.org/wiki/Concurrent_computing>.

[5] "Dining Philosophers Problem." Wikipedia, the free encyclopedia. 20 Apr. 2009 <http://en.wikipedia.org/wiki/Dining_philosophers_problem>.

[6] Meyer, B. 1987. Eiffel: programming for reusability and extendibility. *SIGPLAN Not.* 22, 2 (Feb. 1987), 85-94. DOI= http://doi.acm.org/10.1145/24686.24694

[7] "Eiffel (Programming Language)." Wikipedia, the free encyclopedia. 20 Apr. 2009 <http://en.wikipedia.org/wiki/Eiffel_(programming_language)>.

[8] SCOOP | Scoop. 20 Apr. 2009 <http://scoop.origo.ethz.ch/>.

[9] Compton, Michael J. "SCOOP An Investigation of Concurrency in Eiffel." ANU - ANU College of Engineering and Computer Science - DCS. 22 Apr. 2009 <http://cs.anu.edu.au/~Richard.Walker/eiffel/scoop/mc-thesis.pdf>.

[10] "Why Eiffel Might Be Worth a Second Look || kuro5hin.org." Kuro5hin.org || technology and culture, from the trenches. 20 Apr. 2009 <http://www.kuro5hin.org/story/2006/10/31/20640/115>.

[11] "Parallel Computing." Wikipedia, the free encyclopedia. 23 Apr. 2009 <http://en.wikipedia.org/wiki/Parallel_computing>.




*Alex Laird, Cedarville University, Cedarville, Ohio, 45314*