Eolang Compiler for Simple Cases Technical Report Stage II

HSE Team
hsalekh@hse.ru
HSE
Moscow,Russia

Abstract

Eolang is an Object-Oriented Programming language aimed at realizing the pure concept of object-oriented programming paradigm, in which all components of a program are objects. Eolang's main goal is to prove that fully object-oriented programming is possible not only in books and abstract examples but also in real program code aimed at solving practical problems. The EO programming language is a research and development project that remains in an undeveloped state. Our main objective is to develop the language with more reliable programming methods. In this phase, we present our ideas and solution plan and further develop the Eolang compiler in order to achieve this objective.

Keywords: Elegant Objects, EOlang, Compiler, Java, OOP, Maven, Decoration, Datarization, Duck Typing

1 Introduction

Yegor Bugayenko, the author of the "Elegant Objects" concept underlying the EO language, identifies the following problems of existing Object-Oriented Programming languages [7] :

1. Mutability of data generates side effects, which leads to unpredictable behavior of the program.
2. Using null references to denote the absence of a value.
3. Applying reflection breaks the encapsulation of classes, allowing the developer to use classes in unpredictable scenarios.
4. Inheritance allows you to get rid of visible duplication of code in similar classes, however, making any changes to parent classes only becomes more complicated because of the need for consistency of changes with all inherited classes.

Eolang is designed to curb the above points. This work aims to contribute to the development of the Eolang compiler that will be accessible at GitHub repository (OSS/MIT licence). It is expected to be able to build java classes from Eolang source code. The maven wrapper plugin will be used to initiate compiling at build phase to generate class files, and then packaged by Maven later. The result of such a solution will bring us to answer the question: Is it possible to rewrite Java Cloud web-module with Eolang?

2 Solution Approach

2.1 Why it was important to reform the previous version of compiler

The previous version of the compiler was synthetic and just a text translation of EO. It only simulated the behavior of objects in EO with poor performance and had many objects in memory. Above all, it showed potential difficulty in scalability (e.g., add typing) in case of future development. To address these issues, a new model is proposed for the compiler development.

2.2 Development plan

The development of the compiler is divided into separate parts and covers the following features:

1. Translation of Eolang program code into ".java" files in the Java programming language.

2. Programming languages used in the implementation of the source code of the translator, thus, objects of the standard library, and the runtime library - Java.
3. The parser (as well as the lexer) must be implemented (and augmented) as part of the ANTLR4 solution.
4. The translation of the output code in Java is done through the construction and processing of an intermediate model obtained through parsing XML documents of the stage of parsing a program in EO.
5. The final solution (translator) should be delivered in an easy-to-use form: a Maven plugin [8] in the Maven Central repository [4].

2.3 Eolang Objects Translation

This section describes how the translation of the user-defined Eolang object will be performed.

2.3.1 Naming & Scoping Rules. An Eolang object named "obj" of the global scope is translated into a Java class named "EOobj.java". The prefix "EO" is used as a simple measure to escape naming conflicts with Java code. Eolang objects of local scopes (bound attributes of other objects or anonymous objects passed as arguments during application) are not present as separate Java files. Instead, these are put to the scopes they originate from. So, attributes are named private inner classes, and anonymous objects are anonymous Java classes. This technique makes target code denser and delegates the management of scopes and child-parent hierarchies to the Java Virtual Machine (JVM), not the Eolang runtime. The source program "obj" object is translated into a single "EOobj" class in a single EOobj.java file. No manual naming and class connections management. The transpiler is to compile the source program to the target platform as it is organized originally and let the Java Virtual Machine deal with all the relationships,hierarchies, and scoping mechanisms locally (native).

2.3.2 Target Class Hierarchy. The target class extends org.eolang.core.EOObject.

2.3.3 Constructors. The target class has only one constructor for simplicity. There are two possible cases:

1. If the source class has no free attributes, the resulting constructor has no arguments.

2. If the source class has free attributes, the resulting constructor has all the arguments in the order specified in the source program. All arguments of the constructor are of type "EOObject", so the object does not know the actual type. In the case where free attributes are present, the default constructor (in Java) is disabled.

2.3.4 EO Objects Free Attributes Translation. Free attributes of the source object are translated as follows. Assuming there are these free attributes in the source Eolang object (in the order specified):

1. a
2. b
3. c
4. class
5. he133

For each of them, the following will be performed:

1. Generation of a private final field named EO<attributeName>. "EO" prefix is used to allow using some special names from the source language (e.g., class) that may be a reserve word in the target platform (Java). The field is of type EOObject.

2. Generation of a public method named EO<name>. The method's return type is EOObject. The method's body is just return this.EOattr;.

3. The body of the constructor sets the private EO<attributeName> field to the value of the argument EOattr. The proposed model is fully immutable since a code fragment that applies (or copies) a class can set the free attributes only once through the constructor. Fields holding free attributes are private and final. The default (empty) constructor is disabled. Once set, an attribute cannot be changed either from a user code fragment or from the inside of a class itself.Hence, objects are fully immutable.

However, the model has some weaknesses. First, it cannot handle the partial application mechanism.

The proposed solution allows the “fully-applied-at-once” copying technique only.

Secondly, just as in the original transpiler, the model does not perform type checking of the objects being passed for free attributes binding.

2.3.5 Eolang Objects Bound Attributes Translation. Bound attributes of the source object are translated as follows.

1. For every bound attribute with an arbitrary name such as "attr", a method named "EOattr" is generated. The method returns an object of type EOObject.
2. If the bound attribute is constructed through application operation in the source program, then the target code is placed into the method "EOattr" being generated. In this case the target code is generated as follows:
   a. The method returns a new instance of the object being applied in the source program.
   b. The instance is created through its constructor, where all its parameters are passed in the order of the order specified in the source program.
   c. The evaluation strategy is down to the instance being returned (it decides how to evaluate itself on its own).
3. If the bound attribute is constructed through abstraction operation in the source program, then a private inner class named "EOattr" is generated. The inner class is a subclass of the "EOObject" base class. The target code is generated as follows in this case:
   a. Free attributes of the abstracted object are translated as described in subsection "EO Objects Free Attributes Translation".
   b. Bound attributes of the abstracted object are translated as described in this section.
   c. The method "EOattr" that wraps the "EOattr" private inner class returns a new instance of that class passing all arguments (free attributes) to it in the order specified according to the source program.
   d. The evaluation strategy is implemented via overriding the "getDecoratedObject" standard method (as described in detail in the Dataization section).
   e. To access the parent object, the "getParent" is used from EOObject (it is also overridden inside nested classes). It is done for simplicity and may change in the distant future when the compiler [5] becomes more feature-rich.
   f. When the source program explicitly accesses an attribute (e.g., EOattr) of the parent object (e.g., EOobj) of the attribute (e.g., EOboundAttr), it is explicitly translated to “EOobj.this.EOattr”.

It is important to mention that memory leakage issues that inner classes are blamed for do not affect the proposed model performance compared to the original implementation of the compiler (current compiler) since the latter links all the nested objects (attributes) to their parents. As a further optimization, the transpiler may check if a bound attribute created through abstraction does not rely on the parent’s attributes. In this case, the private nested class may be made static (i.e. not having a reference to its enclosing class). As it was mentioned in Naming & Scoping Rules, bound attributes (these are basically nested objects) are put to the scopes they originate from. So, bound attributes are translated into named private inner classes. This technique makes target code denser and also delegates the management of scopes and child-parent hierarchies to the Java Virtual Machine (JVM), not the Eolang runtime.

2.3.6 Anonymous Objects. The idea behind anonymous objects is simple. Every time an anonymous object is used in the source program (either in application to another object or in decoration), it is translated as an local class of the EOObject class right in the context it is originated from. All the principles of class construction take place in the local class just as in named classes (constructor generation, free and bound attributes translation, implementation of decoration & dataization mechanism, etc.).

2.3.7 Decoration & Dataization. Every user-defined object evaluation technique is implemented via an overridden "getDecoratedObject" standard method. The overridden method constructs the

object's decoratee and delegates the object's own evaluation to the decoratee. The result of the evaluation of the decoratee is then returned as the result of the evaluation of the object itself.

If the decoratee is not present for the source program object, an exception is thrown inside the dataization getData method

2.3.8 Accessing Attributes in the Pseudo-Typed Environment. If the user code fragment utilizes an EOObject instance (and the concrete type is unknown), the Java Reflection API is used to access the attributes of the actual subtype. Since both free (inputs) and bound (outputs) attributes are wrapped as methods (that can have from zero to an arbitrary number of arguments) it is possible to handle access to all kinds of attributes through a uniform technique based on the dynamic method invocation Java Reflection API. The technique works as follows:

1. Ask Reflection API to check if a method with the name EOattr and the necessary signature (with the correct number of arguments) exists. If it does not exist, throw an exception. Otherwise, proceed to step 2.

1.1 If method not found, try to complete Point 1 for the getDecoratedObject().

2. Invoke method passing all the arguments into it in the order specified.

2.3.9 Data Primitives and Runtime Objects. Data primitives and runtime objects also extend the EOObject base class. However, the main idea behind them is to make them as efficient as possible. To do that, data primitives objects are implemented in a more simple way compared to the guidelines of the proposed model. For example, bound attributes of the data primitives objects are implemented through methods only (no class nesting is used).

Another important property of the runtime objects is that they are often atomic and, hence define their evaluation technique without a base on the decoratee as mentioned above. Instead, these dataize to the atomic data or perform a more efficient (semi-eager or eager) evaluation strategy.

2.4 Transformation from XML to Medium (intermediate) model

Some dependencies in the XML files of old version of compiler were noted:

1. All abstractions, no matter their true location in the source "*.eo" file, are moved to the first nesting level. The translator makes the tree flat in terms of abstractions. In this way the translator assigns a name to all abstractions (including anonymous ones).
2. All kinds of bound attributes (i.e. abstraction-based & application-based bound attributes) within EO objects are essentially replaced by application in all cases. So, the application remains as application. And abstraction-based attributes turn into an application that refers to a previously nested object on the first nesting level.

Based on these two properties of the XML input document, it was decided to translate from XML to Medium as follows:

1. Translate all abstractions. They are the first level of XML document nesting. Recursion is not applied at this stage, as abstractions are represented in a completely flat (linear) form.
2. For each abstraction from step 1, all internal applications (i.e. bound attributes) are translated. This process is recursive because of the recursiveness of subtrees that belong to the appendices. After step 2, we have a list (array) of abstractions in the Medium model, each of which has a list of bound-attributes defined by applications, which are also translated into the Medium model.

After that several optimizations are made by the resulting code model structure:

1. Code Model Deflate Operation. This operation returns abstractions, which are "flat" (linear) in previous compiler model, to their places. Thus, the applications which defined abstraction-based attributes get a new field (wrappedAbstraction). Also the applications that referenced local abstraction-based objects get this field.

2. Code Model Scope Correction. This operation corrects the Scope of all the Medium model entities. All applications have file, thus, the abstraction object in which they reside. All abstractions have Scope, thus, the parent abstraction object.The operations outlined above help the intermediate Medium code model become closer to the final model, and move away from the XML model. Both optimizations greatly simplify the final code translation from Medium model to Java.

The google java format [1] and Picocog [2] libraries are taken into consideration in order to deliver cleaner and maintainable code. Also ideas from kotlin [9] will be looked at.

3 Deliverable

EO-lang compiler accessible at Github repository (OSS, MIT), which can build java classes from Eolang source code. The maven wrapper plugin will be used to initiate compiling at the build phase to generate class files, and then packaged with Maven later.

3.1 Runtime Model

Class	Description
EOObject	1.Instantiate an EO object (i.e. make a copy of it via a constructor). 2.Dataize an EO object 3.Set a parent object 4.Access attributes of an object
EOData	Used to store data primitives
EODataObject	A wrapper class to interpret EOData as EOObject
EONoData	A primitive class representing the absence of data

3.2 Mapping EO Entities to Java Code Structures

EO	Java
Abstraction	A plain old Java class extending EOObject. The plain old Java class may also be nested or local.
Application	1.Creating a new class instance 2. Direct method call 3.Calling the method via recursion
Package-scope application	Is wrapped by Abstraction because of the absence of package-levels in Java
A free attribute	A class field that should be set via the constructor
A bound attribute	A method that returns an object
Duck typing	Everything is EOObject. Class fields are accessed via Java Reflection
Object dataization	It is the default for all user-defined classes and relies on the getDecoratedObject()
getDecoratedObject()	To be overridden for evaluation strategy and used to access the attributes of the decoratee object
Anonymous EO objects	Implemented via Local Java Classes Nested class — attribute Local class — anonymous object

3.3 Examples

3.3.1 Sample EO Program (covers almost all language features). :

```
[a b] > rectangle
  a.mul b > area
  [] > perimeter
    2 > a
    mul. > @
      a
      add.
        ^.a
        ^.b
```

3.3.2 0-th level global object rectangle.

```
<o line="5" name="rectangle"
                    original-name="rectangle">
 <o line="5" name="a"/>
   <o line="5" name="b"/>
   <o base=".mul" line="6" method="" name="area">
      <o base="a" line="6" ref="5"/>
      <o base="b" line="6" ref="5"/>
   </o>
   <o base="rectangle$perimeter"
      cut="5"
      line="7"
      name="perimeter"
      ref="7">
      <o as="a" base="a" level="1" ref="5"/>
      <o as="b" base="b" level="1" ref="5"/>
     <o as="area" base="area" level="1" ref="6"/>
   </o>
</o>
```

1. free attribute a
2. free attribute b
3. application-style bound attribute area
4. abstraction-style bound attribute perimeter

3.4 Tests (Execution Time)

3.4.1 Fibonacci. 50-th Fibonacci number [12]: 12586269025
Time: 0m0,191s

3.4.2 Pi digits. Pi Digits till 400000 numbers after dot [13]: 3.141595..
Time: 0m4.023s

3.4.3 Factorial. Factorial of 20 [11]: 2432902008176640000
Time: 0m0.126s

3.4.4 Sum of array elements. First 50 elements (1-50) sum [14]: 1275
Time: 0m0.179s

4 Discussion

4.1 Limitation

4.1.1 Anonymous objects must have no free attributes. In a situation where an object needs to be pass to, for instance, an array.map, it is expected to create an object that has an attribute with output parameters, as opposed to that of the previous runtime model.

This is due to the fact that an instance of the class cannot be created without passing the expected arguments. This happens where anonymous objects are used. In future, it may be possible to resolve this issue at the translator level so that the user code will not be affected by this limitation

4.1.2 Recursion now builds only in a lazy mode. What is impossible now: Use recursive definition inside a greedy construct. Greedy constructs include primitive methods like EOadd, EOmul, etc.

Example:

```
2.mul (recursive (n.sub 1))
```

It impossible now, because the recursion, due to greediness, will never be reduced to the result.

What is possible now: Use greed inside recursive constructs

Example:

```
recursive (n.sub 1) (n.mul 2)
```

4.2 Possibilities for future development of the model

Now, with an intermediate code model in Java (i.e., an object tree built on the Medium model in Java), we can:

4.2.1 Add call optimization. There is a predictable and an unpredictable context.

1. Predictable:

   ```
   rectangle > rec
   ```

   In this case, rec is a type-predictable variable. it is obvious that this is a type of some.package.name.here.EOrectangle. In a predictable context, we can opt out of recursion. Then

   ```
   rec. attr 5
   ```

   instead of rec.

   ```
   _getAttribute ("attr", EOint(5L))
   ```

   will be translated to

   ```
   rec.EOattr(EOint(5L))
   ```

   An increase in speed and even greater connectivity of objects is achieved. As a bonus:

partial validation of the program by the Java compiler is in predictable contexts.

2. Unpredictable context:

```
[a b] > rectangle
```

Here it is not obvious what [a] is. In unpredictable contexts, recursion is continually used. The unpredictable context is still one-access to the input attributes from inside the object. All other contexts are predictable

4.2.2 Add other possible optimization. If it's possible to simplify some object. Example:

```
[] > obj
    [] > @
        [] > @
            [] > @
                5.add 7 > @
```

The above can be simplify to:

```
5. add 7 > obj
```

4.2.3 Make the translator detect recursive call cases.

1. It will try to replace normal recursion from object recursion (when objects are created recursively) to functional recursion (method chain). This will work faster and more efficiently but could also take a long time and resources.
2. Tail recursion, the translator does not only covert into a functional chain, but also converts into a loop. Then the tail recursion in EO will be equivalent to a normal loop but that is even a more complex task than point 1.

5 Conclusion

We have presented the solution approach and the progress. As described in the deliverable section, the product of this work is the new Eolang compiler which is now ready and available at GitHub [3]. The new model (transpiler) is embedded into the pipeline of the maven wrapper plugin. The maven wrapper plugin [6] is used to initiate compiling at the build phase to generate class files from Eolang source code, which are subsequently packaged. As displayed above, the performance of this new model shows a great improvement. Appendix (A.1) shows a sample of generated sources by the new runtime for the fibonacci code.

The proposed solution uses Java instead of XSLT[15] because XSLT is not scalable, too complex to comprehend, and can only transform one text to another or one xml fragment to another[10]. A scalable solution should, perhaps, have code that is clearly analyzable.

In the next phase of this research and development, based on the proposed solution, we are going to develop the Java module from cloudBU by rewriting it in Eolang. During this development we will be able to identify bugs, fix them, and add more features to improve the compiler.

References

[1] 2016. google-java-format. https://github.com/google/google-java-format
[2] 2017. Picocog. https://github.com/ainslec/picocog
[3] 2021. EO-lang compiler for simple cases. https://github.com/HSE-Eolang/eo
[4] 2021. Plugin Developers Centre. https://maven.apache.org/plugin-developers/index.html
[5] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. 2006. *Compilers: Principles, Techniques, and Tools (2nd Edition).* Addison-Wesley Longman Publishing Co., Inc., USA.
[6] Raghuram Bharathan. 2015. *MApache Maven Cookbook.* Packt. https://www.packtpub.com/product/apache-maven-cookbook/9781785286124
[7] Yegor Bugayenko. 2017. Elegant Objects. https://www.elegantobjects.org/
[8] Sonate Company. 2008. *Maven: The Definitive Guide.* O'Reilly Media, Inc. https://www.oreilly.com/library/view/maven-the-definitive/9780596517335/
[9] Dmitry Jemerov and Svetlana Isakova. 2017. Kotlin in Action. https://www.manning.com/books/kotlin-in-action
[10] Michael Kay. 2008. *XSLT 2.0 and XPath 2.0 Programmer's Reference, 4th Edition.* Michael Kay. https://www.wiley.com/en-us/XSLT+2+0+and+XPath+2+0+Programmer%27s+Reference%2C+4th+Edition-p-9780470192740
[11] HSE Team. 2021. Factorial example. https://github.com/HSE-Eolang/eo/blob/master/sandbox/eo/app.eo

[12] HSE Team. 2021. Fibonacci example. https://github.com/HSE-Eolang/eo/blob/master/sandbox/eo/fibonacci.eo
[13] HSE Team. 2021. Pi digits example. https://github.com/HSE-Eolang/sandbox-examples/blob/main/eo/pi.eo
[14] HSE Team. 2021. Sum of array example. https://github.com/HSE-Eolang/sandbox-examples/blob/main/eo/sum.eo
[15] Doug Tidwel. 2008. *XSLT, 2nd Edition.* O'Reilly Media, Inc. https://www.oreilly.com/library/view/xslt-2nd-edition/9780596527211/

A Appendix

A.1 Fibonacci Example

A.1.1 app.eo.

```
+package sandbox
+alias fibonacci sandbox.fibonacci
+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf

[args...] > app
  sprintf > @
    "%dth Fibonacci number is %d\n"
    (args.get 0).toInt > n
    fibonacci n
```

A.1.2 fibonacci.eo.

```
    +package sandbox

[n] > fibonacci
  if. > @
    n.less 3
    small n
    rec n 1 1

  [n] > small
    if. > @
      n.eq 2
      1
      n

  [n minus1 minus2] > rec
    if. > @
      n.eq 3
      minus1.add minus2
      rec (n.sub 1) (minus1.add minus2) minus1
```

A.1.3 EOapp.java.

```
package sandbox;

import org.eolang.*;
import org.eolang.core.*;

/** Package-scope object 'app'. */
public class EOapp extends EOObject {

  /** Field for storing the 'args' variable-length free attribute. */
  private final EOarray EOargs;

  /**
```

```
   * Constructs (via one-time-full application) the package-scope object 'app'.
   *
   * @param EOargs the object to bind to the 'args' variable-length free attribute.
   */
  public EOapp(EOObject... EOargs) {
    this.EOargs = new EOarray(EOargs);
  }

  /** The decoratee of this object. */
  @Override
  public EOObject _getDecoratedObject() {
    return new org.eolang.txt.EOsprintf(
        new EOstring("%dth Fibonacci number is %d\n"),
        this.EOn(),
        new sandbox.EOfibonacci(this.EOn()));
  }

  /** Returns the object bound to the 'args' input attribute. */
  public EOarray EOargs() {
    return this.EOargs;
  }

  /** Application-based bound attribute object 'n' */
  public EOObject EOn() {
    return ((this.EOargs())._getAttribute("EOget", new EOint(0L)))._getAttribute("EOtoInt");
  }
}
```

A.1.4 EOfibonacci.java.

```
package sandbox;

import org.eolang.*;
import org.eolang.core.*;

/** Package-scope object 'fibonacci'. */
public class EOfibonacci extends EOObject {

  /** Field for storing the 'n' free attribute. */
  private final EOObject EOn;

  /**
   * Constructs (via one-time-full application) the package-scope object 'fibonacci'.
   *
   * @param EOn the object to bind to the 'n' free attribute.
   */
  public EOfibonacci(EOObject EOn) {
    this.EOn = EOn;
  }

  /** The decoratee of this object. */
  @Override
  public EOObject _getDecoratedObject() {
    return ((this.EOn())._getAttribute("EOless", new EOint(3L)))
        ._getAttribute(
```

```
          "EOif", this.EOsmall(this.EOn()), this.EOrec(this.EOn(), new EOint(1L), new EOint(1L)));
  }

  /** Returns the object bound to the 'n' input attribute. */
  public EOObject EOn() {
    return this.EOn;
  }

  /** Abstraction-based bound attribute object 'small' */
  public EOObject EOsmall(EOObject EOn) {
    return new EOsmall(EOn);
  }

  /** Abstraction-based bound attribute object 'rec' */
  public EOObject EOrec(EOObject EOn, EOObject EOminus1, EOObject EOminus2) {
    return new EOrec(EOn, EOminus1, EOminus2);
  }

  /** Attribute object 'small' of the package-scope object 'fibonacci'. */
  private class EOsmall extends EOObject {

    /** Field for storing the 'n' free attribute. */
    private final EOObject EOn;

    /**
     * Constructs (via one-time-full application) the attribute object 'small' of the package-scope
     * object 'fibonacci'.
     *
     * @param EOn the object to bind to the 'n' free attribute.
     */
    public EOsmall(EOObject EOn) {
      this.EOn = EOn;
    }

    /** Returns the parent object 'fibonacci' of this object */
    @Override
    public EOObject _getParentObject() {
      return EOfibonacci.this;
    }

    /** The decoratee of this object. */
    @Override
    public EOObject _getDecoratedObject() {
      return ((this.EOn())._getAttribute("EOeq", new EOint(2L)))
          ._getAttribute("EOif", new EOint(1L), this.EOn());
    }

    /** Returns the object bound to the 'n' input attribute. */
    public EOObject EOn() {
      return this.EOn;
    }
  }

  /** Attribute object 'rec' of the package-scope object 'fibonacci'. */
```

```
private class EOrec extends EOObject {

  /** Field for storing the 'n' free attribute. */
  private final EOObject EOn;
  /** Field for storing the 'minus1' free attribute. */
  private final EOObject EOminus1;
  /** Field for storing the 'minus2' free attribute. */
  private final EOObject EOminus2;

  /**
   * Constructs (via one-time-full application) the attribute object 'rec' of the package-scope
   * object 'fibonacci'.
   *
   * @param EOn the object to bind to the 'n' free attribute.
   * @param EOminus1 the object to bind to the 'minus1' free attribute.
   * @param EOminus2 the object to bind to the 'minus2' free attribute.
   */
  public EOrec(EOObject EOn, EOObject EOminus1, EOObject EOminus2) {
    this.EOn = EOn;
    this.EOminus1 = EOminus1;
    this.EOminus2 = EOminus2;
  }

  /** Returns the parent object 'fibonacci' of this object */
  @Override
  public EOObject _getParentObject() {
    return EOfibonacci.this;
  }

  /** The decoratee of this object. */
  @Override
  public EOObject _getDecoratedObject() {
    return ((this.EOn())._getAttribute("EOeq", new EOint(3L)))
        ._getAttribute(
            "EOif",
            (this.EOminus1())._getAttribute("EOadd", this.EOminus2()),
            new EOrec(
                (this.EOn())._getAttribute("EOsub", new EOint(1L)),
                (this.EOminus1())._getAttribute("EOadd", this.EOminus2()),
                this.EOminus1()));
  }

  /** Returns the object bound to the 'n' input attribute. */
  public EOObject EOn() {
    return this.EOn;
  }

  /** Returns the object bound to the 'minus1' input attribute. */
  public EOObject EOminus1() {
    return this.EOminus1;
  }

  /** Returns the object bound to the 'minus2' input attribute. */
  public EOObject EOminus2() {
```

```
      return this.EOminus2;
    }
  }
}
```