

Genetic Programming meets Model-Driven Development

Thomas Weise, Michael Zapf, Mohammad Ullah Khan, Kurt Geihs
Distributed Systems Group, University of Kassel
Wilhelmshöher Allee 73, 34121 Kassel, Germany
weise | zapf | khan | geihs@vs.uni-kassel.de

Preview

This document is a preview version and not necessarily identical with the original.

Abstract

Genetic programming is known to provide good solutions for many problems like the evolution of network protocols and distributed algorithms. Then, it is most likely a hardwired module of a design framework where it assists the engineer in optimizing specific aspects in system development. In this paper we show how the utility of genetic programming can be increased remarkably by isolating it as a component and integrating it into the model-driven software development process. Our genetic programming framework produces XMI-encoded UML models that can easily be loaded into widely available modeling tools, which in turn offer code generation as well as additional analysis and test capabilities. We use the evolution of a distributed election algorithm as an example to illustrate how genetic programming can be combined with model-driven development.

1 Introduction

Genetic programming is the automated generation of computer programs by artificial evolution. Among many other applications, it has successfully been applied in the area of distributed computing for evolving proactive aggregation protocols [1] and in transforming global to local behavior [2]. In that research, it was integrated as a fixed component into a software system, as is the case in most of the other current projects. As a hardwired module, it provides its results through an application-internal interface or at least in a format that usually can only be used by exactly this system. Especially for the evolution of algorithms,

such a restriction makes no sense. Algorithms are general, platform-independent descriptions of processes. It would be more reasonable if they were returned in an independent format.

In this paper we discuss how the results of genetic programming can be represented in a standardized format which allows them to be analyzed, transformed, and tested.

2 Genetic Programming and MDD

In genetic programming, the genome and often even the phenotypic representations of the solution candidates differs from the format in which the results will be delivered. Algorithms for example are often evolved in a tree-like form. They do not need to be compiled but can be interpreted directly, since the instructions in the AST can be executed on simple virtual machines. Delivered are the algorithms however in C code for instance.

Such a translation of the internal representation into the desired output format takes place in many applications of genetic programming. The disadvantage of this direct transformation is the fixed binding of the phenotypes to one single applicable representation. Translating the results into an intermediate format which can be processed by different tools would add great flexibility while only requiring little additional work.

2.1 Model-Driven Development

Model-Driven Development (MDD) [3] and the Model-Driven Architecture (MDA) [4] guided by the Object Management Group (OMG) are key technologies for software and system design. Developing applications starts with an elaborate modeling phase. Although the times of writing programs from scratch have long been gone, there are few software engineering methodologies that impose such clear and formal guidelines like MDD. The model is a simple and intuitive specification of the application and can be transformed into program code step-by-step. At the beginning, a platform-independent model (PIM) is created which only

describes the semantics, abstracting from a specific technology. The PIM is transformed into a more fine-grained, platform-specific model (PSM) bound to a certain target system. Finally, the PSM is transformed to source code in a programming language. These steps can be performed by automated tools which reduces programming errors and decreases development costs.

MDA recommends using the Unified Modeling Language (UML) [5] for model specification, the most popular and wide-spread modeling language in software technology. The syntax of UML is defined by the UML meta-model which in turn is an instance of the Meta-Object Facility (MOF) [6]. MOF models can be exchanged between different MDD tools in the standardized XML Metadata Interchange format (XMI) [7]. For UML models a special schema exists, allowing them to be serialized as XMI.

2.2 Combining MDD and GP

After translating the results of genetic programming into XMI-encoded UML models, they can be imported into a wide range of MDD tools and we can use their code generation and transformation abilities to translate the models into implementations for a variety of target platforms. We additionally can perform further optimizations, tests, and analyses on the algorithms using standard software engineering methods.

Genetic programming will most often not create whole applications. Instead, it will just evolve certain functionality which can be encapsulated in a module. It is quite possible that the application which will include this module is modeled in UML. Then the algorithms can be integrated directly into its model, achieving a consistent view of the whole system in one common specification.

It also becomes much easier to combine different (evolved) algorithms. One example for such a situation would be that an algorithm is needed which transmits messages along the spanning tree of a sensor network a specific event is detected. It is very unlikely that one could genetically evolve an algorithm that is able to perform this task as a whole. Thus, a divide-and-conquer strategy should be applied. An algorithm could be evolved that automatically finds and maintains a spanning tree and another algorithm can be grown that optimizes the detection of the event. If both results are returned as XMI, they can be combined in a software design tool with very little effort.

3 Evolving Distributed Algorithms

Today we experience a growing demand for *sensor networks*. Sensor networks are composed of a large number of sensor nodes, small devices that gather information about

their environment and transmit it wirelessly. They are restricted in resources like memory size, processing speed, and – most importantly – battery power. Distributed algorithms for sensor nodes should thus be as energy-efficient as possible.

Simulating a single node does not suffice to evaluate the fitness of such algorithms, hence we simulate whole sensor networks. In our simulation, sensor nodes are represented as virtual machines. As in reality, many nodes run asynchronously in the simulation at approximately the same speed, which may differ from node to node and cannot be assumed to be constant. Transmissions are broadcasted like radio waves that spread into all directions and are received by any node in range.

We apply multi-objective genetic programming since it allows us to optimize the algorithms for different aspects. As functional objective, we perform a comparison of the observed behavior of the simulated network (running the evolved algorithms) with the desired global behavior. The evolutionary algorithm hence transforms global behavior of a network into local behavior of single nodes, thus effectively creating emergence. Non-functional objective functions foster the economical use of resources, especially for minimizing energy-expensive communication.

3.1 Evolving an Election Algorithm

Election means to select one node out of a group of nodes whereby at the end all nodes should have knowledge of the ID of this special node. In distributed systems, an election is performed when e.g. a node is needed to act as a communication relay or as a coordinator. For the purpose of this example, we assume that the active node with the maximum ID shall be selected.

In the simulations, we initialize all virtual machines with their own ID in the first memory cell. If an algorithm makes progress, the nodes should have assumed greater (valid) IDs after some time. A fully functional algorithm would accomplish that the first memory cells of all nodes contain the maximum ID.

One simple result obtained is displayed in Figure 1. It lets nodes broadcast the greatest ID encountered in a loop, reducing network traffic by performing dummy work. The algorithm consists of two parts: a procedure that is called when the node starts up (`procedure_0`) and an asynchronously called, interrupt-like routine which receives incoming messages (`procedure_1`).

4 Creating a PIM

The example algorithm introduced in the previous section has been evolved as a tree of instructions. The next step is to transform it into a suitable UML model and to

```

called on startup procedure_0
store 1st variable into output buffer
0: push mem[0]
1: some useless operations used
2: to delay and, as a consequence,
3: reduce transmissions in the
4: simulated/evaluated time span
send output buffer
go back to start
5: send
6: goto 0

called asynchronously when a message comes in
compare the known and the received value
if no improvement then exit
exchange values
procedure_1
0: zf = (params[0] < mem[0])
1: if zf then goto 3 //≡ exit
2: xchg params[0], mem[0]

```

Figure 1. One of the non-dominated solutions.

create XMI-formatted output subsequently used as input for source code creation.

We have to analyze which entities must be specified in order to describe an algorithm completely. In principle, each algorithm is constituted by three parts:

1. the data structures the algorithm uses,
2. the primitive instructions that work on these data structures, and
3. the control flow (i.e. the sequence of primitive instructions).

4.1 Control Flow Model

The control flow of an algorithm can easily be represented using an activity diagram. In general, the single procedures of an algorithm are modeled as compound activities including a set of simple actions. Each of them corresponds either to the execution of a single instruction or to a branch to another procedure. Transitions define the sequence of the instructions inside the procedures and denote unconditional jumps whereas conditional jumps are represented by decision nodes.

Since we evolve distributed algorithms, we also need means for modeling transmissions. Broadcasting a message corresponds to sending a signal and the asynchronous mechanism of receiving messages is modeled as a parallel thread. This thread contains an infinite loop of a receive event action followed by a call to the message handler. Figure 2 illustrates the control flow of the example algorithm from Section 3.1.

4.2 Data Model

The nodes of a distributed system as well as the virtual machines used for simulation can be regarded as instances of a class and are therefore modeled in a class diagram. They have a fixed-size memory `mem`, a stack `stack` and a

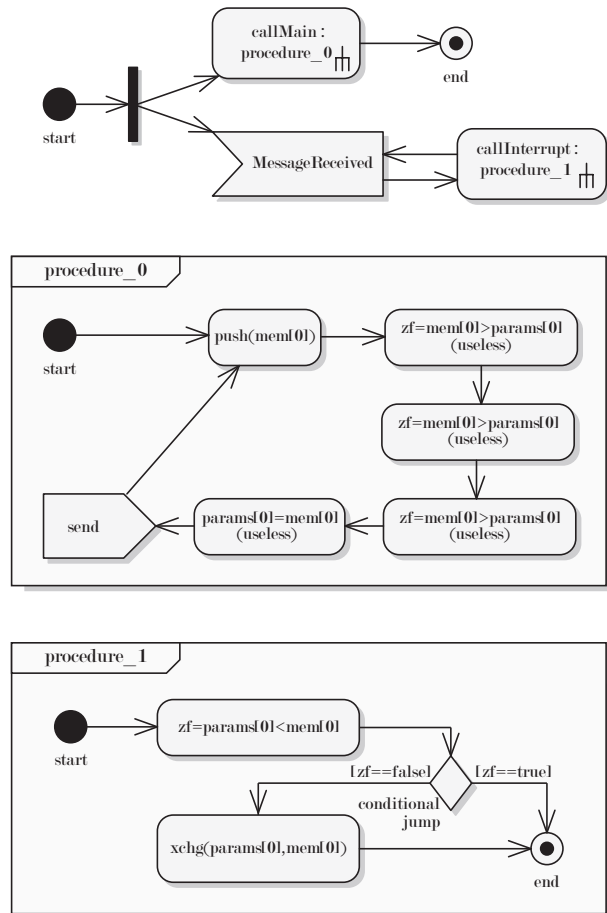


Figure 2. The control flow of the evolved election algorithm.

flag register `zf`. Parameters for procedures are stored in an additional parameter list `params`. The interrupt-like procedure invoked whenever a message comes in finds the message stored in this array. These data structures are member variables of the node class.

4.3 Modeling the Primitive Operations

Finally, the primitive operations used by the algorithms must be specified. The pseudo code used in Figure 1 contains three different types of operations:

1. Operations that modify the control flow like `goto` or procedure calls are already defined in the activity diagrams (see Figure 2) – as transitions, decision nodes, or behavior-invoking actions.
2. Operations with obvious semantics like value assignments or arithmetic operations require no additional specification.
3. The semantics of operations like pushing something onto

the stack (`push`) or transmitting all data currently on the stack as message (`send`) need to be defined more precisely. They can be specified as post-conditions using the Object Constraint Language OCL [8].

5 Transforming the UML Models

In our work, we support such automatic source code generation using the *MOFScript* language, a model-to-text transformation language obtained from the MODELWARE project [9]. At present, MOFScript supports transforming UML models created using an Eclipse Modeling Framework implementation of a subset of OMG UML 2.x meta-model, obtained from the *Eclipse UML2 Project* [10].

In the following, we show a C source code fragment generated solely using MOFScript transformations. Due to page limitations we cannot show the MOF script sources. The code below corresponds to the model of `procedure_1` in Figure 2, `procedure_0` has been omitted.

```

1 int[] mem, stack, params;
2 bool zf; int stackPtr;
3 ...
4 void procedure_0() {
5     a0: push(mem[0]);           goto a1;
6     a1: zf = mem[0] > params[0]; goto a2;
7     a2: zf = mem[0] > params[0]; goto a3;
8     a3: zf = mem[0] > params[0]; goto a4;
9     a4: params[0] = mem[0];     goto a5;
10    a5: send();                 goto a0;
11    a6:; }
12
13 void procedure_1() {
14    a0: zf = params[0] < mem[0]; goto a1;
15    a1: if(zf) goto a3; else goto a2;
16    a2: xchg(params, 0, mem, 0); goto a3;
17    a3:; }

```

The MOFScript transformation to Java is slightly more complex since we need to emulate absolute jumps.

```

1 public class Node {
2     private final int[] mem;
3     private int[] params;
4     private boolean zf;
5     ...
6     public void procedure_1() {
7         int ip;
8         for (ip = 0; ip < 3;) {
9             switch (ip) {
10                case 0: { zf = params[0] < mem[0];
11                        ip = 1; break; }
12                case 1: { if (zf) ip = 3;
13                        else ip = 2;
14                        break; }
15                case 2: { xchg(params, 0, mem, 0);
16                        ip = 3; break; }
17            } }
18    } }

```

6 Conclusions

The goal of our work is to prove the utility of genetic programming as a tool for developing distributed algorithms.

Finding cases where genetic programming can assist in creating distributed systems is, however, only a first step. It is likewise important to incorporate its results into the application development.

In this paper we have shown that the utility of genetic programming can considerably be enhanced by integrating it into the model-driven development process. Furthermore, we have demonstrated that such integration can be accomplished using existing and widely available tools. We will now assess the performance of the output of the tool chain introduced here on a real sensor network platform.

A more elaborate discussion of the topic of this paper can be found in [11].

References

- [1] Thomas Weise, Kurt Geihs, and Philipp Andreas Baer. Genetic programming for proactive aggregation protocols. In *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms ICANNGA'07, Part 1*, pages 167–173, 2007. See proceedings [12]. Online available at <http://www.it-weise.de/documents/files/W2007DGPFb.pdf> (version September 17, 2007) and http://www.springerlink.com/content/978-3-540-71589-4/?p_o=10 (version 2007-08-13).
- [2] Thomas Weise and Kurt Geihs. Dgpf – an adaptable framework for distributed multi-objective search algorithms applied to the genetic programming of sensor networks. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, pages 157–166, 2006. See proceedings [13]. Online available at <http://www.it-weise.de/documents/files/W2006DGPFc.pdf> (version September 17, 2007).
- [3] Jan Oyvind Aagedal, Jean Bezivin, and Peter F. Lintington. Model-driven development. In J. Malenfant and Bjarte M. Ostvold, editors, *ECOOP 2004 Workshop Reader, Proceedings of the ECOOP 2004 Workshop*, volume 3344 of *LNCS*. Springer-Verlag, Oslo, Norway, June 14-18 2004. Published in January 2005.
- [4] Uwe Abmann, Mehmet Aksit, and Arend Rensink, editors. *Model Driven Architecture, European MDA Workshops: Foundations and Applications, MDFA 2003 and MDFA 2004*, Twente, The Netherlands,

June 26-27, 2003 and Linköping, Sweden, June 10-11, 2004, *Revised Selected Papers*, volume 3599 of *Lecture Notes in Computer Science*, ISBN: 3-540-28240-8. Springer, 2005.

- [5] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, third edition, ISBN: 0-3211-9368-7, 978-0201657838, September 15 2003.
- [6] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. Model comparison: a foundation for model composition and model transformation testing. In *GaMMA '06: Proceedings of the 2006 international workshop on Global integrated model management*, pages 13–20, New York, NY, USA, Shanghai, China, 2006, ISBN: 1-5959-3410-3. ACM Press. Online available at <http://doi.acm.org/10.1145/1138304.1138308> (version 2007-09-17).
- [7] International Organization for Standardization. *ISO/IEC 19503:2005-11*, 2005.
- [8] Heinrich Hußmann and Steffen Zschaler. The object constraint language for UML 2.0 – overview and assessment. *Upgrade*, 5, April 2004.
- [9] Arend Rensink and Jos Warmer, editors. *Proceedings of the Second European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA 2006*, volume 4066 of *Lecture Notes in Computer Science (LNCS)*, ISBN: 3-540-35909-5, ISSN: 0302-9743 (Print) 1611-3349 (Online). Springer Berlin/Heidelberg, Bilbao, Spain, July 10-13 2006.
- [10] Anna Gerber and Kerry Raymond. Mof to emf: there and back again. In *eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 60–64, New York, NY, USA, Anaheim, California, October 17 2003. ACM Press. Online available at <http://sky.fit.qut.edu.au/~raymondk/MOF-to-EMF-There-and-Back-Again.pdf> (version 2007-09-17).
- [11] Thomas Weise, Michael Zapf, Mohammad Ullah Khan, and Kurt Geihs. Genetic programming meets model-driven development. Research Paper 2007, 2, University of Kassel, July 2 2007. Persistent Identifier: urn:nbn:de:hebis:34-2007070218786. Online available at <http://www.it-weise.de/documents/index.html?pubonly\#WZKG2007DGPfd> (version 2007-09-17) and <http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007070218786> (version 2007-09-17).
- [12] Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors. *Proceedings of Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Part I*, volume 4431/2007 of *Lecture Notes in Computer Science (LNCS)*, ISBN: 978-3-540-71589-4, ISSN: 0302-9743. Springer Berlin Heidelberg New York, Warsaw University of Technology, Warsaw, Poland, April 11-14 2007. see also [14], <http://icannga07.ee.pw.edu.pl/> (version 2007-08-31), and <http://www.springerlink.com/content/978-3-540-71590-0/> (version 2007-08-31).
- [13] Bogdan Filipič and Jurij Šilc, editors. *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, ISBN: 978-961-6303-81-1. Jožef Stefan Institute, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, October 9-10 2006.
- [14] Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors. *Proceedings of Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Part II*, volume 4432/2007 of *Lecture Notes in Computer Science (LNCS)*, ISBN: 978-3-540-71590-0, ISSN: 0302-9743. Springer Berlin Heidelberg New York, Warsaw University of Technology, Warsaw, Poland, April 11-14 2007. see also [12], <http://icannga07.ee.pw.edu.pl/> (version 2007-08-31), and <http://www.springerlink.com/content/978-3-540-71589-4/> (version 2007-08-31).

```
@inproceedings{WZKG2007DGPFG,
author      = {Thomas Weise and Michael Zapf and Mohammad Ullah Khan and
              Kurt Geihs},
title       = {Genetic Programming meets Model-Driven Development},
booktitle   = {Proceedings of Seventh International Conference on Hybrid Intelligent
              Systems},
editor      = {Andreas K{\"}nig and Mario K{\"}ppen and Ajith Abraham and Christian
              Igel and Nikola Kasabov},
ISBN        = {0-7695-2946-1},
publisher   = {IEEE Computer Society},
year        = {2007},
month       = sep # {-18},
location    = {Kaiserslautern, Germany},
note        = {Conference website: http://his07.hybridsystem.com/, Library of Congress
              Number 2007936727, Product Number E2946\
              The work is online available at
              http://www.it-weise.de/documents/index.html\#WZKG2007DGPFG. \
              The paper can be downloaded at
              http://www.it-weise.de/documents/files/WZKG2007DGPFG.pdf. \
              The poster can be downloaded at
              http://www.it-weise.de/documents/files/WZKG2007DGPFG\\_poster.pdf. \
              Contact Thomas Weise at tweise@gmx.de or http://www.it-weise.de/.},
abstract    = {Genetic programming is known to provide good solutions for many problems
              like the evolution of network protocols and distributed algorithms. In
              such cases it is most likely a hardware module of a design framework that
              assists the engineer to optimize specific aspects of the system to be
              developed. It provides its results in a fixed format through an internal
              interface. In this paper we show how the utility of genetic programming
              can be increased remarkably by isolating it as a component and integrating
              it into the model-driven software development process. Our genetic
              programming framework produces XML-encoded UML models that can easily be
              loaded into widely available modeling tools which in turn possess code
              generation as well as additional analysis and test capabilities. We use
              the evolution of a distributed election algorithm as an example to
              illustrate how genetic programming can be combined with model-driven
              development. This example clearly illustrates the advantages of our
              approach - the generation of source code in different programming
              languages.},
contents    = {* Introduction\
              * Genetic Programming and MDD\
              - Model-Driven Development\
              - Combining MDD and GP\
              * Evolving Distributed Algorithms\
              - Evolving an Election Algorithm\
              * Creating a PIM\
              - Control Flow Model\
              - Data Model\
              - Modeling the Primitive Operations\
              * Transforming the UML Models\
              * Conclusion},
keywords    = {Genetic Programming, GP, Model Driven Development, MDD, Model Driven
              Architecture, MDA, XML, MOF-Skript, UML, Distributed Algorithms},
url         = {http://www.it-weise.de/documents/index.html\#WZKG2007DGPFG}
}
```