

International Journal of Computational Intelligence and Applications
© World Scientific Publishing Company

COMBINING GENETIC PROGRAMMING AND MODEL-DRIVEN DEVELOPMENT

THOMAS WEISE, MICHAEL ZAPF, MOHAMMAD ULLAH KHAN, KURT GEIHS

*University of Kassel, Distributed Systems Group
34121 Kassel, Germany*

*<http://www.vs.uni-kassel.de>
{weise/zapf/khan/geihs}@vs.uni.kassel.de*

Received (received date)

Revised (revised date)

Genetic Programming is known to provide good solutions for many problems like the evolution of network protocols and distributed algorithms. In most cases it is a hardwired module of a design framework assisting the engineer in optimizing specific aspects in system development. In this article 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.

Keywords: Genetic programming; evolutionary algorithms; model-driven development; model-driven architecture; UML; XMI; MOFScript; distributed systems; distributed algorithms; sensor networks.

1. Introduction

Genetic Programming (GP) can be defined as the automated generation of computer programs by artificial evolution. Among many other applications, it has successfully been used for creating protocols with minimum communication costs[16, 52, 10], software testing[14], and for evolving hardware/software co-designs[13]. In our previous work, we applied Genetic Programming in the area of distributed computing.[47] We evolved proactive aggregation protocols for large-scale distributed systems and local algorithms that create a specified global network behavior[45, 46].

In this research, Genetic Programming utilizes an internal algorithm representation especially tailored for evolution and simulation. Like in many of the current GP systems, this internal representation is then translated to one specific programming language. If this translation is performed automatically by an output filter of the system, we speak of a *monolithic* GP application^a.

^aWe use the same terminology for systems that evolve individuals directly in the target representation like Grammar-guided Genetic Programming.

2 *Weise, Zapf, Khan, Geihs*

Especially for the evolution of algorithms, such a restriction to one programming language makes no sense. Algorithms are general, platform-independent descriptions of processes. It would be more reasonable if they were returned in an independent format by the Genetic Programming system.

In this article 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 usually also the phenotypic representations of the solution candidates differ 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 on simple virtual machines. The result of the evolution, *C* code for instance, is produced by a transformation filter processing the output of the Genetic Programming system, as illustrated in Figure 1.

Such a translation of the internal representation into the desired output format takes place in many applications of Genetic Programming. Its disadvantage 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 (see Figure 2) while only requiring little additional work.

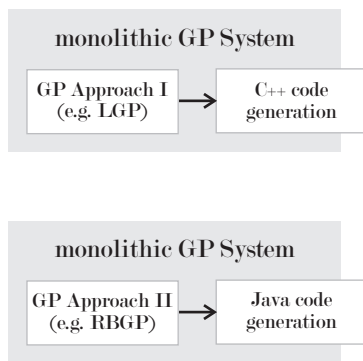


Fig. 1. Monolithic GP

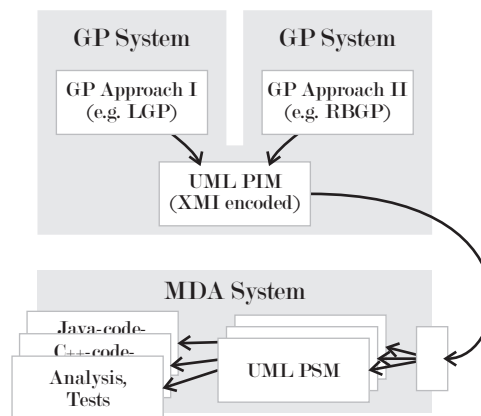


Fig. 2. GP with XMI output MDA

2.1. Model-Driven Development

Model-Driven Development[3] (MDD) and the Model-Driven Architecture[4, 39] (MDA) guided by the Object Management Group (OMG) are key methodologies

for software and system design. They suggest to start the application development with an elaborate modeling phase. Although the times of writing programs from scratch have long been gone, there are few software engineering technologies that impose such clear and formal guidelines like MDA. A model is a simple and intuitive specification of the application and can be transformed into program code.

Initially, a platform-independent model (PIM) is created which only describes the application semantics, abstracting from a specific technology. The PIM can be transformed into a platform-specific model (PSM) bound to a certain target system. Finally, the PSM is transformed to source code in a programming language^b. These steps can be performed by automated tools which prevent programming errors, speed up the development process, and thus decrease the software development costs. There exist various methods to perform model transformations, spanning from QVT[25] (Query, View and Transformation, model-to-model transformation), MOFScript[33] (model-to-source code) to direct XML transformations using XSLT style sheets.

MDA recommends using the Unified Modeling Language[18] (UML) for model specification, the most popular and wide-spread modeling language in software engineering. The model elements of UML are defined by the UML meta-model which in turn is an instance of the Meta-Object Facility[27] (MOF). MOF models can be exchanged between different MDD tools in the standardized XML Metadata Interchange format[23] (XMI). There exists a schema for UML models that allows 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 can additionally perform further optimizations, tests, and analyses on the algorithms using standard software engineering methods.

Genetic Programming will usually not create whole applications. Instead, it will just evolve certain functionality which can be encapsulated in a module. If the application which will include this module is also modeled in UML, the grown algorithms can be integrated directly into its model. This way we can achieve a consistent view of the whole system in one common specification.

It also becomes much easier to combine different (evolved) program parts. One example for such a situation would be that an algorithm is needed which transmits messages along the spanning tree of a sensor network if a specific event is detected. It is 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

^bIn some cases, the PSM itself is already source code. In these cases, the model-to-text transformation is applied to the PIM.

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, maybe by using different, problem-specific internal representations. If both results are returned as XMI, they can be combined in a software design tool with very little effort.

3. Related Work

In the last two decades, the area of automated protocol generation using evolutionary algorithms has been visited by different researchers. Yamaguchi et al. have proposed a method to derive protocol specifications from service specifications which as described as Petri Net models with registers[51]. Based on this idea, they formulate and solve a message exchange optimization problem in order to reduce communication costs[16]. Tschudin has developed a new execution model for computer communications called *Fraglets*[42]. Yamamoto and he were able to evolve Fraglet-based protocols for reliable and unreliable communication channels[52]. Special applications like efficient broadcasting on different network topologies were the subject of the studies of Comellas and Giménez[10].

The evolution of distributed algorithms itself is indeed a new area. One important perspective on this topic is given by Qureshi[36, 38] who discusses the evolution of agents. He showed that communication behavior can be evolved along with agent cooperation and that it is possible that agents can “learn” how to communicate with other predefined agents by Genetic Programming. Offline emergence engineering with GP for distributed and communicating agent societies is one of our recent research activities. We were able to show that load balancing algorithms based on multi-agent systems can be evolved[53].

Currently, a lot of research is done on the software engineering sector in order to improve the model-driven development approach[3, 49]. The translation of the application models to program code is a vital part of MDD tool chains. Many modeling tools like Enterprise Architect, Poseidon, and IBM Rational Software Architect/Modeller therefore provide some template facilities for this code generation. Although these components are already matured, defining and coordinating the appropriate transformations is still a cumbersome task and there is often the need for subsequent manual code insertion.

A more attractive approach adopted by many works[7, 29, 11, 24] is to export the models to the XMI interchange format and then using XSLT transformations to produce the source code. Here, the drawback is that the style sheets will only work with one specific version of XMI and also require in-depth knowledge this format. MOFScript[33], on the other hand, provides the advantage of defining the transformation script from the perspective of the UML model, with all the benefits, abstractions, and direct access to the interdependencies among the modeling elements[1, 34].

A similar approach is practiced in the MDWorkbench[40] system, where rules

and sets of procedural expressions for querying model elements can be defined. Within a rule, text templates can be called to generate output. These templates are specified in the Text Generation Language (TGL) and can produce both, static and dynamic texts, depending on the model information. These aspects are integrated into MOFScript directly and there is no need to invoke external templates. This makes it a very light-weight tool which yet can produce flexible text output.

With its current state of maturity, MDD gains value for applications with more scientific character like the design of multi-agent[35] or real-time systems[41]. No research however has yet been conducted on the integration of Genetic Programming into the model-driven software development process itself.

4. Evolving Distributed Algorithms

As already mentioned in the introduction, the focus of our work is put on utilizing Genetic Programming for distributed systems. In this section we give an introduction on how a desired behavior of a network can be translated by Genetic Programming into algorithms that run on its nodes.[45] After elaborating on the key aspects of this topic, we give a small example which we will solve with two different GP approaches and which is used later for demonstration purposes.

An important aspect of the evolution of distributed algorithms is how the objective values, describing the utility of the evolved programs, are determined. In our experiments, we therefore use simulation environments for whole networks. The nodes of the network are represented by virtual machines. As in reality, many of them run asynchronously at approximately the same speed, which may differ from node to node and cannot be assumed to be constant either. 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. Non-functional objective functions foster the economical use of resources, minimizing communication and program size, for instance.

4.1. Evolving an Election Algorithm

Election algorithms have many applications in distributed systems. In Windows domains they are used to select the master browser[15], and in many group communication protocols the coordinator is determined with an election[43]. They are part of clock synchronization methods as well[20]. Election is also a very common functionality needed in numerous agent scenarios[12].

A distributed election algorithm can be initiated by any number of nodes in the system and will reach a terminal configuration in which exactly one node is elected as *leader* and all nodes agree to this choice[31]. The challenge of such an

election procedure is the distributed nature: It must be ensured that all nodes get a consistent view of the election.

In the simulations, we initialize all virtual machines with their own ID in a dedicated memory cell or variable. 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. If the algorithm is also resource-friendly, it should reach this goal with as few transmissions as possible.

Therefore, we apply three objective functions: the first function, subject to maximization, is the aggregation of all valid IDs stored in the first memory cells of the nodes over all time steps (see equation Equation 1).

$$f_1 = \sum_{\forall \text{ time } t} \sum_{\forall \text{ nodes } n} \begin{cases} n.mem[0](t) & \text{if } valid(n.mem[0](t)) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

It is an indicator both for the functionality as well as the convergence speed of the evolved algorithms. The second and third objective functions both optimize non-functional aspects: They are used to minimize the number of messages sent and the length of the algorithms.

4.2. *Using Linear Genetic Programming*

In contrast to the traditional tree-based Genetic Programming methods, linear Genetic Programming (LGP) evolves algorithms as sequences of instructions, very much like assembler code [32, 8, 9].

One of the results obtained with linear Genetic Programming guided by the aforementioned objective functions is displayed in Figure 3. 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 (an automatically defined function, ADF) which receives incoming messages (`procedure_1`). In this simple algorithm, the nodes constantly broadcast the greatest ID they have encountered in a loop, reducing the network traffic only by performing dummy work. By constantly sending (probably unnecessary) messages and thus not being optimal, the algorithm ensures that nodes started later will still take part in the election.

4.3. *Using Rule-Based Genetic Programming*

As already mentioned, one of the advantages of our approach is that we can use the same XMI output converter to produce PIM models for different Genetic Programming methods. We have already discussed the above linear Genetic Programming solution in our previous work[50, 49]. This time we apply the output conversion to an election algorithm grown using Rule-based Genetic Programming[48, 53] (RBGP).

Rule-based Genetic Programming is very different from linear Genetic Programming in that it has been developed with the goal of reducing epistasis[48]. In terms

```

called on startup procedure_0
store 1st variable into 0: push mem[0]
output buffer 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 5: send
go back to start 6: goto 0
-----
called asynchronously when procedure_1
a message comes in 0: zf = (params[0] < mem[0])
compare the known and 1: if zf then goto 3 //≡ exit
the received value 2: xchg params[0], mem[0]
if no improvement then exit
exchange values

```

Fig. 3. An election algorithm evolved using LGP

```

(int > at) ∨ (at ≤ 0) ⇒ send
(0 > 1) ∨ (at ≤ 0) ⇒ outt+1 = idt
<< useless >>
true ∧ (at < int) ⇒ at+1 = int
<< useless >>
(at > 1) ∨ (at < int) ⇒ outt+1 = int

```

Listing 1. An election algorithm evolved using RBGP

of evolutionary algorithms, epistasis identifies unwanted dependencies between different parts of a genotype. Changing, for instance, one part of a program has severe impact on the meaning of the other parts. Furthermore, the order in which the instructions of a program are executed is very important. Hence, epistasis increases the problem hardness for evolutionary algorithms.

In RBGP, a program is defined as a set of rules which are executed in a cycle where the condition parts of each rule are checked. If a condition evaluates to `true`, the corresponding action part (on the right side of the rule) is executed. Unlike in LGP, where the memory is directly represented as an array of cells, RBGP programs work on a set of named variables. Write access to a variable is performed to a temporary storage and committed after all rules are evaluated (this is why the symbols in Listing 1 are annotated with the subscripts t and $t+1$). As a result, the order of the rules plays no role anymore and the epistasis is effectively reduced.

In the election algorithm experiment with RBGP, three special variables are defined: `id` contains the id of node, `out` contains the values to be sent to other nodes, and `in` will be filled with the contents of incoming messages. A message (containing the value of `out`) is transmitted when the command `send` is invoked. `a` is a multi-purpose variable which is expected to be filled with the result by the algorithm. Listing 1 depicts an RBGP program denoting an evolved election algorithm.

5. Creating a PIM

These algorithms have both been evolved in internal representations – exactly as displayed in Figure 3 and Listing 1. The next step is to transform them into suitable UML models and to produce XMI-formatted output which will be used as input for creating source code.

First we have to analyze which entities must be specified in order to describe a (distributed) algorithm completely. In principle, each algorithm is constituted by three parts:

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

5.1. *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 4 illustrates the control flow of the example algorithm from Section 4.1.

5.2. *Data Model*

The nodes of a distributed system as well as the virtual machines that we use to simulate our evolved algorithms can be modeled in a class diagram. They have a fixed-size memory, a stack and a flag variable. Parameters for procedures (also integer numbers) are stored in an additional parameter list. The interrupt-like procedure, which is invoked whenever a message comes in, finds the message stored in this array. These data structures can be modeled as member variables: the memory `mem`, the parameter array `params` and the stack `stack` are lists of integer numbers, whereas the flag `zf` is a Boolean value.

It should be noted that modeling nodes as classes does not necessarily imply the application of object-oriented programming. Classes are just means to specify data structures along with operations working on them. Figure 5 shows the class diagram of our example algorithms.

5.3. *Modeling the Primitive Operations*

Finally, the primitive operations used by the algorithms must be specified. Generally, our distributed algorithms can consist of three different types of operations:

- (1) Operations that modify the control flow like jumps (`goto` in the LGP example) or procedure calls are already defined in the activity diagram of Section 5.1 – as transitions, decision nodes, or behavior-invoking actions.
- (2) Operations with obvious semantics like value assignments or arithmetic operations require no additional specification.

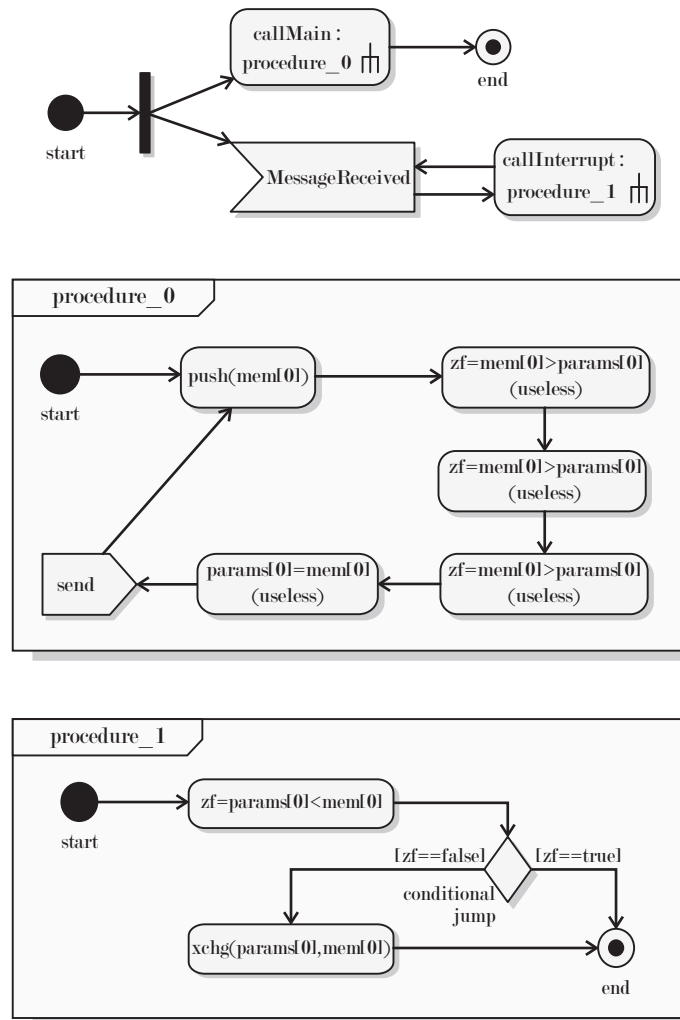


Fig. 4. The control flow of the evolved LGP election algorithm from Figure 3.

- (3) Other operations like pushing something onto the stack or transmitting all data currently on the stack as message (**push** and **send** in the LGP example) need to be defined more precisely. Their semantics can be specified as post-conditions using the Object Constraint Language OCL[22].

5.4. Modeling LGP and RBGP Programs

The modeling language discussed above has first been derived for imperative programs such as the results of linear Genetic Programming. Hence, algorithms as displayed in Figure 3 can be directly transformed into platform-independent models

Node
- mem: int[0..n]
- params: int[0..n]
- stack: int[0..10]
- zf: boolean
+ pop(): int
+ procedure_0(): void
+ procedure_1(): void
+ push(int): void
+ send(): void

Fig. 5. The data structure definition of the evolved election algorithm.

like the one illustrated in Figure 4 on the preceding page.

Of course, the introduced primitives and structures also apply to programs expressed in the rule-language of RBGP. In RBGP, the number of variables is fixed, so we can map each variable to a single cell in `mem`. Additionally, we use the `params` array as temporary storage for the symbols that have been modified. The code that commits this storage is automatically attached to the control flow model. The interrupt called whenever messages come in is no longer needed since incoming messages are always of length 0 and are still written to the memory address representing the symbol `in` in the `params` array. The `send` action in RBGP can be expressed as a `send` command in the code flow model with a preceding `push` of the memory cell that stands for the `out` symbol (see also Listing 3). An RBGP program now is an infinite loop which iterates over (nested) decision nodes representing the single rules.

Therefore, these models are larger than those produced by translating LGP programs and have not been illustrated here for this reason. However, they can be transformed using exactly the same means as can be applied to the models of LGP programs and that are discussed in the following.

The important point is that the expense for creating an XMI-output writer only needs to be spent once (at least per problem domain), since this component can be reused for translating different internal representations into UML models.

6. Transforming the UML Models

One of the goals of the MDD-based development is the possibility to automatically generate code from the application models using transformation tools. The models provided as output of our GP system are basically platform-independent. They can either be translated to platform-specific models first or directly to source code in a given programming language. In our work, we have chosen the latter. We support such automatic source code generation using the *MOFScript* language, a model-to-text transformation language obtained from the MODELWARE project[39]. MOF-

Script is currently a candidate in the OMG RFP process on MOF Model-to-Text Transformation[33] and is intended to cover the aspects required in the context of text generation in software engineering.

At present, MOFScript supports transforming UML models created using an EMF-based^c implementation of a subset of OMG UML 2.x metamodel, obtained from the *Eclipse UML2 Project*[19].

```

...
int[] mem;
int[] stack;
int stackPtr;
int[] params;
bool zf;

...
void procedure_0() {
    a0: push(mem[0]);          goto a1;
    a1: zf = mem[0] > params[0]; goto a2;
    a2: zf = mem[0] > params[0]; goto a3;
    a3: zf = mem[0] > params[0]; goto a4;
    a4: params[0] = mem[0];    goto a5;
    a5: send();               goto a0;
    a6:;
}

void procedure_1() {
    a0: zf = params[0] < mem[0]; goto a1;
    a1: if(zf) goto a3;
        else goto a2;
    a2: xchg(params, 0, mem, 0); goto a3;
    a3:;
}

```

Listing 2. Figure 3 translated to C++

```

public class Node2 {
    private final int[] mem;
    private final int[] stack;
    private int stackPtr;
    private int[] params;
    private boolean zf;

    ...

    public void procedure_0() {
        int ip;
        for (ip = 0; ip < 33;) {
            switch (ip) {
                case 0: { param[0] = mem[0];
                        ip++; }
                    ...
                case 4: { zf = mem[1] > mem[0];
                        ip = 5;
                        break; }
                case 5: { if (zf) ip = 8;
                        else ip = 6;
                        break; }
                case 6: { zf = mem[0] < 0;
                        ip++;
                        break; }
                case 7: { if (zf) ip = 8;
                        else ip = 9;
                        break; }
                case 8: { push(mem[2]);
                        send();
                        ip++;
                        break; }
                    ...
                case 31: { mem[0] = params[0];
                        ip++; }
                case 32: { mem[1] = params[1];
                        ip = 0; }
                    ...
            }
        }
    }
}

```

Listing 3. Listing 1 translated to Java

The translation of the model (Figure 4) of the election algorithm of Figure 3 grown with LGP outlined in Listing 2 is very short and efficient. During the transformation, different actions are identified as *Activity nodes* and are given numbers,

^cEclipse Modeling Framework

starting from 0. In `procedure_1` which is called whenever a message is received, the action `zf=params[0]>mem[0]` is denoted as action 0. The conditional jump gets number 1 and `xchg(params[0], mem[0])` becomes action 2, while the initial action is not numbered. For the transformation to *C++*, we now can store all actions as a sequence in which each action is labeled with its number. After an instruction has been executed, a `goto` performs a jump to the next one. Since compilers will usually omit useless jumps, the resulting code is very efficient.

In *Java*, no jump instructions are available. For demonstration purposes, we simply emulate them with an iterated switch-case construct. For example, if we transform the (more extensive) model of Figure 3 evolved with RBGP to Java, the result will look like Listing 3. This loop always starts with the number of the action that executes right at the beginning of the procedure. After each action, the control flow transcends to the next one by assigning its number to the `ip` variable, until a value greater or equal to the total number of actions is selected which will lead to the termination of the loop (which does not happen in RBGP). In Listing 3, we can see the already discussed translation of the RBGP `send` action to its more general pendant using a buffer by prepending `push`. The second extension of the evolved control flow adds instructions at the front that copy the current state of the variables in `mem` to the temporary storage `params` which is later modified by the rules and committed at the end.

The transformation itself is, of course, independent from such modifications of the models and can be applied for any valid UML model based on our considerations in Section 5. The resulting code may not look as structured as if written by a human being, but it is as same as functional.

7. Conclusions

In this paper we have shown that the utility of Genetic Programming can be remarkably enhanced by integrating it into the model-driven development process. By providing one XMI output component and attaching it to different GP systems, we become able to use MOFScript transformations for creating source code in different programming languages. We have demonstrated that such integration can be accomplished by using existing and widely available tools. There are also lots of possible additional benefits of combining Genetic Programming and MDA.

One important problem in programming is that it is generally not possible to prove the correctness of a program by testing. The computation of the objective values in GP is a form of testing (using simulation environments). In other words, the algorithms produced by Genetic Programming are not necessarily correct – they just behave adequately in all test scenarios [53, 45]. A model-driven development tool chain however may provide additional validation and checking utilities which can help to decrease the chance of hidden errors in the evolved programs to remain undiscovered.

In the future we will perform research mainly in two directions, specifically

extending the use of Genetic Programming to other domains in the context of distributed systems as well as enhancing its integration into the application development process according to current software engineering practices and tools.

References

1. Model to text transformation in practice: Generating code from rich associations specifications. In J. F. Roddick, R. Benjamins, S. S.-S. Cherfi, R. Chiang, R. Elmasri, H. Han, M. Hepp, M. Lystras, V. Misic, G. Poels, I.-Y. Song, J. Trujillo, and C. Vangenot, editors, *Advances in Conceptual Modeling – Theory and Practice. Proceedings of the ER 2006 Workshops BP-UML, CoMoGIS, COSS, ECDM, OIS, QoIS, SemWAT*, volume 4231/2006 of *Lecture Notes in Computer Science (LNCS)*, pages 63–72. Springer Berlin/Heidelberg, Nov. 6–9, 2006.
2. *Proceedings of BIONETICS 2007, 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), IEEE, ACM, Dec. 10–13, 2007.
3. J. O. Aagedal, J. Bezivin, and P. F. Linington. Model-driven development. In J. Malenfant and B. M. Ostvold, editors, *ECOOP 2004 Workshop Reader, Proceedings of the ECOOP 2004 Workshop*, volume 3344 of *LNCS*. Springer-Verlag, June 14–18, 2004. Published in January 2005.
4. U. Aßmann, M. Aksit, and A. 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 (LNCS)*. Springer, 2005.
5. B. Beliczyński, A. Dzieliński, M. Iwanowski, and B. 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)*. Springer Berlin Heidelberg New York, Apr. 11–14, 2007. See also [6], <http://icannga07.ee.pw.edu.pl/> [accessed 2007-08-31], and <http://www.springerlink.com/content/978-3-540-71590-0/> [accessed 2007-08-31].
6. B. Beliczyński, A. Dzieliński, M. Iwanowski, and B. 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)*. Springer Berlin Heidelberg New York, Apr. 11–14, 2007. See also [5], <http://icannga07.ee.pw.edu.pl/> [accessed 2007-08-31], and <http://www.springerlink.com/content/978-3-540-71589-4/> [accessed 2007-08-31].
7. J. Bézin. From object composition to model transformation with the mda. In *Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems*

14 REFERENCES

- (*TOOLS39*), pages 350–354, July 29–Aug. 3, 2001. Online available at <http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/TOOLS.USA.pdf> [accessed 2008-02-19].
8. M. F. Brameier. *On Linear Genetic Programming*. PhD thesis, Fachbereich Informatik, Universität Dortmund, Feb. 2004. Day of Submission: 2003-05-28, Committee: Wolfgang Banzhaf and Martin Riedmiller and Peter Nordin. Online available at <https://eldorado.uni-dortmund.de/handle/2003/20098> and <http://hdl.handle.net/2003/20098> [accessed 2007-08-17].
 9. M. F. Brameier and W. Banzhaf. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines*, 2(4):381–407, Dec. 2001. Online available at <http://dx.doi.org/10.1023/A:1012978805372> [accessed 2007-09-09].
 10. F. Comellas and G. Giménez. Genetic programming to design communication algorithms for parallel architectures. *Parallel Processing Letters (PPL)*, 8(4):549–560, Dec. 1998. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Comellas_1998_GPD.html and <http://citeseer.ist.psu.edu/comellas98genetic.html> [accessed 2007-09-14].
 11. A. D’Ambrogio. A model transformation framework for the automated building of performance models from uml models. In *Proceedings of the 5th international workshop on Software and performance*, pages 75–86, July 12–14, 2005. Online available at <http://doi.acm.org/10.1145/1071021.1071029> [accessed 2008-02-19].
 12. S. Das, P. Flocchini, A. Nayak, and N. Santoro. Effective elections for anonymous mobile agents. In *Algorithms and Computation*, volume 4288/2006 of *Lecture Notes in Computer Science (LNCS)*, chapter Session 9B: Distributed Computing and Cryptography, pages 732–743. Springer Berlin/Heidelberg, 2006.
 13. M. N. de Miranda, R. N. B. Lima, A. C. P. Pedroza, and A. C. de Mesquita. Hw/sw codesign of protocols based on performance optimization using genetic algorithms. In J. M. D. Souza, N. L. S. da Fonseca, and E. A. S. e Silva, editors, *Teletraffic Engineering in the Internet Era: Proceedings of the International Teletraffic Congress*, volume 1 of *Teletraffic Science and Engineering*, volume 4, pages 259–269. Amsterdam: Elsevier, North-Holland Publishing Co, Sept. 2001. Online available at <http://citeseer.ist.psu.edu/553319.html> and <http://www.gta.ufrj.br/ftp/gta/TechReports/MBAM01b.ps.gz> [accessed 2009-09-13].
 14. A. Derezińska. Advanced mutation operators applicable in c# programs. In K. Sacha, editor, *Software Engineering Techniques: Design for Quality – IFIP Working Conference on Software Engineering Techniques - SET 2006*, pages 283–288. Springer, Oct. 17–20, 2006. Co-located with VIII Conference on Software Engineering – KKIO 2006.
 15. R. Eckstein, D. Collier-Brown, and P. Kelly. *Using Samba*. O’Reilly & Associates, Inc., 1st edition, Nov. 1999. Order Number: 4495. Online available at

- <http://www.oreilly.com/catalog/samba/chapter/book/> [accessed 2007-10-24].
16. K. El-Fakihi, H. Yamaguchi, and G. von Bochmann. A method and a genetic algorithm for deriving protocols for distributed applications with minimum communication cost. In *Proceedings of Eleventh IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'99)*, Nov. 3–6, 1999. Online available at <http://citeseer.ist.psu.edu/430349.html> and <http://www-higashi.ist.osaka-u.ac.jp/~h-yamagu/resource/pdcs99.pdf> [accessed 2007-09-14].
 17. B. Filipič and J. Šilc, editors. *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)*. Jožef Stefan Institute, Oct. 9–10, 2006. Online available at http://is.ijs.si/is/is2006/Proceedings/C/BIOMA06_Complete.pdf [accessed 2008-11-05]. Part of the Information Society Multiconference (IS 2006).
 18. M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, third edition, Sept. 15, 2003.
 19. A. Gerber and K. 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, Oct. 17, 2003. ACM Press. Online available at <http://sky.fit.qut.edu.au/~raymondk/MOF-to-EMF-There-and-Back-Again.pdf> [accessed 2007-09-17].
 20. R. Gusella and S. Zatti. An election algorithm for a distributed clock synchronization program. Technical Report UCB/CSD-86-275, EECS Department, University of California, Berkeley, 1986. Online available at <http://www.eecs.berkeley.edu/Pubs/TechRpts/1986/CSD-86-275.pdf> [accessed 2007-10-24].
 21. A. Hartman and D. Kreische, editors. *Proceedings of the First European Conference on Model Driven Architecture – Foundations and Applications, ECMDA-FA 2005*, volume 3748 of *Lecture Notes in Computer Science (LNCS)*. Springer, Nov. 7–10, 2005.
 22. H. Hußmann and S. Zschaler. The object constraint language for UML 2.0 – overview and assessment. *Upgrade*, 5(2), Apr. 2004.
 23. International Organization for Standardization. *ISO/IEC 19503:2005-11*, 2005.
 24. S. L. Jim. From uml diagrams to behavioural source code. Master's thesis, Universiteit van Amsterdam, Sept. 2006. Online available at <http://homepages.cwi.nl/~paulk/thesesMasterSoftwareEngineering/2006/SabrinaJim.pdf> [accessed 2007-09-17].
 25. S. R. Judson, D. L. Carver, and R. B. France. A metamodeling approach to model transformation. In *OOPSLA'03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 326–327, New York, NY, USA, 2003. ACM Press. Online available at <http://doi.acm.org/10.1145/949344.949435> [accessed 2007-09-16].
 26. K. E. Kinneer, Jr., editor. *Advances in Genetic Programming*, volume 1 of *Com-*

16 REFERENCES

- plex Adaptive Systems*. MIT Press, Cambridge, MA, USA, Apr. 7, 1994. Partly online available at <http://books.google.com/books?id=eu2Jp1nQdBkC> [accessed 2008-09-16].
27. D. S. Kolovos, R. F. Paige, and F. A. 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, 2006. ACM Press. Online available at <http://doi.acm.org/10.1145/1138304.1138308> [accessed 2007-09-17].
 28. A. König, M. Köppen, A. Abraham, C. Igel, and N. Kasabov, editors. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*. IEEE Computer Society, Sept. 17–19, 2007. Product Number E2946. see <http://his07.hybridsystem.com/> [accessed 2007-09-01].
 29. J. Kovse and T. Härder. Generic xmi-based uml model transformations. In *Proceedings of the 8th International Conference on Object-Oriented Information Systems*, pages 192–198, Sept. 2–5, 2002. Online available at <http://wwdvs.informatik.uni-kl.de/pubs/papers/KH02.00IS.pdf> and <http://whitepapers.silicon.com/0,39024759,60093303p,00.htm> [accessed 2008-02-19].
 30. J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors. *Proceedings of the First Annual Conference Genetic Programming (GP-96)*. MIT Press, July 28–31, 1996.
 31. G. Le Lann. Distributed systems – towards a formal approach. In B. Gilchrist, editor, *Information Processing, Proceedings of International Federation for Information Processing World Computer Congress, IFIP Congress 77*, pages 155–160, Amsterdam, The Netherlands, Aug. 8–12, 1977. North Holland / Elsevier Science.
 32. P. Nordin. A compiling genetic programming system that directly manipulates the machine code. In *Advances in genetic programming 1*, chapter 14, pages 311–331. MIT Press, 1994. In collection [26]. Machine code GP Sun Spark and i868.
 33. J. Oldevik, T. Neple, R. Grønmo, J. Aagedal, and P. Desfray. Second revised submission for MOF model to text transformation language RFP, Nov. 2005. developed by SINTEF in the European IP project 511731 MODELWARE.
 34. J. Oldevik, T. Neple, R. Grønmo, J. Ø. Aagedal, and A.-J. Berre. Toward standardised model to text transformations. In *ECMDA-FA, Proceedings of Model Driven Architecture – Foundations and Applications*, pages 239–253, 2005. In proceedings [21].
 35. J. Pavón, J. J. Gómez-Sanz, and R. Fuentes. Model driven development of multi-agent systems. In *ECMDA-FA Proceedings of Model Driven Architecture – Foundations and Applications*, pages 284–298, 2006. In proceedings [39].
 36. M. A. Qureshi. Evolving agents. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 369–374, 1996. In proceedings [30] and also [37]. Online available at

- http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/AQ_gp96.ps.gz
[accessed 2007-09-17].
37. M. A. Qureshi. Evolving agents. Research Note RN/96/4, UCL, Gower Street, London, WC1E 6BT, UK, Jan. 1996. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/AQ_gp96.ps.gz and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/qureshi_1996_eaRN.html [accessed 2008-09-02]. See also [36].
 38. M. A. Qureshi. *The Evolution of Agents*. PhD thesis, University College, London, London, UK, July 2001. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/qureshi_thesis.html and <http://citeseer.ist.psu.edu/759376.html> [accessed 2007-09-14]. Supervisor: Jon Crowcroft.
 39. A. Rensink and J. 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)*. Springer Berlin/Heidelberg, July 10–13, 2006.
 40. S. SAS. Mdworkbench, 2007. See <http://www.mdworkbench.com/> [accessed 2008-02-19].
 41. D. C. Schmidt. Model driven development for distributed real-time and embedded systems. In *MoDELS, Proceedings of Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005*, volume 3713 of *Lecture Notes in Computer Science (LNCS)*, page 1. Springer, Oct. 2–7, 2005.
 42. C. F. Tschudin. Fraglets – a metabolistic execution model for communication protocols. In *Proceedings of 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS)*, June 30–July 3, 2003. Online available at <http://cn.cs.unibas.ch/pub/doc/2003-ains.pdf> and <http://path.berkeley.edu/ains/final/007%20-%2022-tschudin.pdf> [accessed 2008-05-02]. See also <http://www.fraglets.net/> [accessed 2007-09-17].
 43. S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. UMass Computer Science Technical Report 03-20, Computer networks and performance evaluation research group. Computer Science Department, University of Massachusetts, 2004. Online available at ftp://gaia.cs.umass.edu/pub/Vasu03_LdrElec.pdf [accessed 2007-10-24]. See also [44].
 44. S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proceedings of the 12th IEEE International Conference on Network Protocols, IEEE ICNP 2004*, pages 350–360, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331, USA, Oct. 5–8, 2004. IEEE Computer Society. IEEE Computer Society Order Number P2161. Online available at <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.957> and ftp://gaia.cs.umass.edu/pub/Vasu03_LdrElec.ps.gz [accessed 2008-11-16] and

18 REFERENCES

- online available at ftp://gaia.cs.umass.edu/pub/Vasudevan04_icnp.pdf [accessed 2007-10-24]. See also [43].
45. T. Weise and K. 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. In proceedings [17]. Online available at <http://www.it-weise.de/documents/files/W2006DGPFc.pdf> [accessed 2009-04-18].
 46. T. Weise and K. Geihs. Genetic Programming Techniques for Sensor Networks. In *Proceedings of 5. GI/ITG KuVS Fachgespräch “Drahtlose Sensornetze”*, pages 21–25, July 17–18, 2006. Online available at <http://www.it-weise.de/documents/files/W2006DGPFb.pdf> and <http://elib.uni-stuttgart.de/opus/volltexte/2006/2838/> [accessed 2007-11-07].
 47. T. Weise, K. Geihs, and P. A. 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. In proceedings [5]. Online available at <http://www.it-weise.de/documents/files/W2007DGPFb.pdf> [accessed 2009-04-18].
 48. T. Weise, M. Zapf, and K. Geihs. Rule-based Genetic Programming. In *Proceedings of BIONETICS 2007, 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2007. In proceedings [2]. Online available at <http://www.it-weise.de/documents/files/WZG2007REBP.pdf> [accessed 2009-04-18].
 49. T. Weise, M. Zapf, M. U. Khan, and K. Geihs. Genetic Programming meets Model-Driven Development. In *Proceedings of Seventh International Conference on Hybrid Intelligent Systems*, 2007. In proceedings [28]. Online available at <http://www.it-weise.de/documents/files/WZKG2007DGPFg.pdf> [accessed 2009-04-18].
 50. T. Weise, M. Zapf, M. U. Khan, and K. Geihs. Genetic Programming meets Model-Driven Development. *Kasseler Informatikschriften (KIS) 2007, 2*, University of Kassel, July 2, 2007. Online available at <http://www.it-weise.de/documents/files/WZKG2007DGPFd.pdf> and <http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007070218786> [accessed 2007-09-17].
 51. H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi. Synthesis of protocol entities’ specifications from service specifications in a petri net model with registers. In *ICDCS’95: Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 510–517. IEEE Computer Society, Washington, DC, USA, May 30–June 2, 1995. Online available

- at <http://citeseer.ist.psu.edu/yamaguchi95synthesis.html> [accessed 2007-11-19].
52. L. A. R. Yamamoto and C. F. Tschudin. Genetic evolution of protocol implementations and configurations. In *IFIP/IEEE International workshop on Self-Managed Systems and Services (SelfMan 2005)*, May 19, 2005. Online available at <http://cn.cs.unibas.ch/pub/doc/2005-selfman.pdf> [accessed 2007-09-17].
 53. M. Zapf and T. Weise. Offline Emergence Engineering For Agent Societies. In *Proceedings of the Fifth European Workshop on Multi-Agent Systems (EUMAS'07)*, Dec. 14, 2007. Also presented at the co-located Fifth Technical Forum Group (TFG5). Online available at <http://www.it-weise.de/documents/files/ZW2007EUMASTR.pdf> [accessed 2009-04-18]. See also [54].
 54. M. Zapf and T. Weise. Offline Emergence Engineering For Agent Societies. Kasseler Informatikschriften (KIS) 2007, 8, University of Kassel, FB16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, Dec. 7, 2007. Online available at <https://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007120719844> and <http://www.it-weise.de/documents/files/ZW2007EUMASTR.pdf> [accessed 2007-11-20], see also [53].

20 REFERENCES

```
@article{WZKG2009DGPfz,  
  author      = {Thomas Weise and Michael Zapf and Mohammad Ullah Khan  
                and Kurt Geihs},  
  title       = {{Combining Genetic Programming and Model-Driven  
                Development}},  
  journal     = {International Journal of Computational Intelligence  
                and Applications (IJCIA)},  
  affiliation = {University of Kassel, FB-16, Distributed Systems Group,  
                Wilhelmsh{"o}her Allee 73, 34121 Kassel, Germany},  
  issn        = {1469-0268, 1757-58850},  
  month       = mar,  
  year        = {2009},  
  volume      = {8},  
  number      = {3},  
  language    = {en},  
  keywords    = {Genetic Programming, GP, Model Driven Development,  
                MDD, Model Driven Architecture, MDA, XMI, MOF-Skript, UML,  
                Distributed Algorithms},  
  publisher   = {World Scientific -- Connecting Great Minds},  
  editor      = {Brijesh Verma},  
  url         = {http://www.it-weise.de/documents/files/WZKG2009DGPfz.pdf},  
}
```