SolidOpt – Innovative Multiple Model Software Optimization Framework

Vassil G. Vassilev*
FMI, University of Plovdiv
"Paisii Hilendarski"

Alexander P. Penev†
FMI, University of Plovdiv
"Paisii Hilendarski"

Tihomir K. Petrov‡
FMI, University of Plovdiv
"Paisii Hilendarski"

ABSTRACT

This paper discusses building up a framework, which helps the automated and dynamic optimization of the software. Disadvantages of existing frameworks and systems are described. We determine the underlying main ideas and principles in the development of SolidOpt framework. The advantages and the need of using multiple models are discussed. We suggest a scheme, which generalizes the multiple model ideas. Software processes and their interaction with the (operational) environment are analyzed. We show the significance of the environment for the optimal execution of the computer programs. We examine an evolutionary approach for software system optimization depending on its behaviour in the environment. Adaptability and permissibility of the system evolution are discussed. Preliminary results of the experimental framework development are shown. The possibility of using SolidOpt with educational purposes to increase the students' creativity and to encourage them to think innovatively is exposed.

KEYWORDS: software optimization, optimization, software evolution, multi-model architecture, model optimality evolution, IL/bytecode engineering, refactoring, adaptation.

INDEX TERMS: D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement–Restructuring, reverse engineering, and reengineering; D.2.3 [Software Engineering]: Coding Tools and Techniques–Object-oriented programming; D.3.4 [Programming Languages]: Processors–Optimization

1 INTRODUCTION

It is difficult to achieve system execution optimum (optimality) at some point of view (performance, for example), especially when developing large software systems. There are techniques for improving the system optimality in different directions and at different time of its life cycle. Almost every technique needs an interactive intervention of the system developers to introduce the corrections.

If we analyze the problem in detail, it becomes obvious that there are many factors, which influence the system optimality. Many software applications try to solve certain problems in this field. However, most of them are concentrated on concrete development stages of the system.

Creating a framework, which contains elements for software system analysis, is not a novelty. There are frameworks, which contain elements for software system transformation (especially optimization), too. Creating a combination between the two kinds would enable the application to perform focused, automated code transformations. As a result, a mechanism of more accurate optimality control during the entire system life cycle will be given.

Software optimizing represents performance improvement of the target application (or in the general case – the use of fewer resources of the computer system).

The increase in the performance of the computer systems is done mainly by two approaches:

- Hardware – changing the hardware of the computer system with up-to-date and highly productive components. Hardware optimization of the computer system is relatively easy to be realized but it is constrained by the laws of nature;
- Software – creating more optimal computer programs or optimizing the existing programs.

The software optimization is the subject of our discussion.

There are two kinds of software optimization, depending on the way of applying them:

- Manual – very labour-intensive work. It includes collecting information of how the system works. Continuous tests for the efficiency and correctness of the changes should be made. Most of these optimizations are done according to one criterion. It is difficult to consider too many parameters, to seek eventual correlation between them, and to test and evaluate the accomplished result;
- Automated – includes applying diverse optimizing methods from another software system upon the target program. The methods suppose preliminary proof for the semantic equivalency of the transformed program (like the translators, for instance).

During the program compilation the compiler produces executable code. The executable code is stored

*86 Gen. Kartsov Str, Karlovo, BULGARIA, e-mail: vassil_vassilev@hotmail.com
†236 Bulgaria Blvd, Plovdiv, BULGARIA, e-mail: apenev@uni-plovdiv.bg
‡2 Gen. Nikola Riazkov Str, Gabrovo, BULGARIA, e-mail: petrov.dev@gmail.com

and it is unnecessary to be compiled again. It should be mentioned that compilers do not always produce optimal executable code. When compiling large systems, the production of "unoptimal" code reflects negatively on the system's overall performance. There is a need for creating modules inside the compilers (including the JIT-compilers), which analyze different representations of the source code and apply preliminary determined optimizations.

The main disadvantages of the build-in optimizations in the translators are their:

- Static nature – applied once at compile time;
- General nature – specific information for the domain is not used;
- "Built-in" in the translator – cannot be easily extended by the target system developers.

Besides, the application of the methods in JIT-compilers is constrained by a requirement for performance.

Some of the most often used tools for analyzing software systems are FxCop [1], StyleCop [2], Gendarme [3], FindBugs [4], PMD [5], Bandera [6], JLint [7][8], ESC/Java [9]. For further comparison between some of them see [10].

Gendarme and FxCop are extensible rule-based tools designed to find problems in .NET applications and libraries. They inspect programs and libraries that contain code in ECMA CIL format (Mono and .NET) and search for common problems with the code, problems that compilers do not usually check or have not historically checked.

FindBugs is a bug pattern detector similar to Gandrame. This tool analyzes Java bytecode and performs syntactic checks and data flow analysis on program source code. FindBugs uses a series of ad hoc techniques designed to balance precision, efficiency, and usability. One of the main techniques used by FindBugs is to match syntactically source code to known suspicious programming practices.

Bandera uses a completely different technique. To use Bandera, the programmers either annotate their source code with specifications describing what should be checked, or do not provide specifications in case they only want to verify some standard synchronization properties. In the latter case, Bandera verifies the absence of deadlocks. Bandera includes optional slicing and abstraction phases, followed by model checking.

ATOM (Analysis Tools with OM) [11] is a single framework for building a wide range of customized program analysing tools. The user simply defines the tool-specific details in instrumentation and analysis routines. Building a basic block counting tool with ATOM requires only a page of code.

Vulcan [12] extends the main ideas of ATOM. It performs both static and dynamic code modifications in heterogeneous and distributed software systems.

BARBER (Binary Refactoring browser for Java) [13] is a tool for bytecode transformations such as Split Class, Glue Class, Inline/Devirtualize Method, Remove Delegate and Remove Visitor.

Soot [14] is a framework for optimizing Java bytecode[14]. The framework is implemented in Java and supports three intermediate models for representing Java bytecode: Baf, Jimple, and Grimp.

Phoenix [15] is the codename of a framework for software analysis and optimization, which will be the foundation of the future Microsoft compilers. It is an extendible system, which could be adjusted for reading and writing binary and MSIL assemblies. Phoenix represents the input files in IR (Intermediate Representation), which can be analyzed and manipulated by using the so-called Phoenix API. Many of the ideas are fundamental but they are oriented towards building compilers, which limits the use of the framework.

The above-described systems are modern but they do not support a unified mechanism for analysing, profiling and optimizing a wide range of arbitrary applications. Some of them work for concrete programming languages. Others are concentrated only on analysing or transforming the software systems, or cover only a part of the entire system life cycle (development time, for instance).

Creating such a mechanism would simplify the problem of finding the optimal execution values and parameters of a given software system.

Our main objective is to create a framework, which contains utilities for analysis and automated transformation of software systems. After its final development stage it will help to create:

- Optimizing-friendly software systems – systems, which ensure easy application of optimization methods upon themselves (their design and implementation are done with the knowledge of the framework existence);
- Tools – they will be used for optimizing during the entire life cycle of the system.

The framework supposes several user roles:

- Developer role – the developer uses the framework during programming to achieve optimality (with the exclusive knowledge of the framework and the program model);
- Integrator role – the integrator uses tools based on the framework to optimize implemented programs (the integrator does not have knowledge of the program model and the concept of the program);
- End user role – the end user employs a program written by the framework or tools, based on the framework, and makes extra settings on the optimizing profiles. Statistics during the execution can be used for extra personalization of the optimization techniques.

Transformation methods of the framework use the information, obtained from the analyzing tools (static or dynamic) to perform automated transformations.

The operation of such systems can be represented by architectural pattern, which is discussed further.

2 SOFTWARE AS A MODEL

Programming languages are notations for describing computations to people and to machines. The world as we know it depends on programming languages, because all the software running on all the computers has been written in some programming language [24]. Software can be treated as a prescription, solving concrete real-world problems. Every prescription or algorithm, which describes part of the real world, is a model. We came to the conclusion that a software program is a (execution) model for solving a given problem.

Within time the program model should become more and more abstract and closer to the natural language, i.e. to turn into a model that is clearer to the developer. This leads to the need of applications, reducing to a low-level model that is executable from a concrete virtual execution system (VES). The process (known as translation) leads to the loss of information about the high-level model and particularly to the generation of "unoptimal" code for execution. The "unoptimal" translation is a result of three main reasons:

- Missing or not very well implemented optimizing modules in the translator;
- Not using the full capabilities of the VES;
- Not using the whole information, available in the high-level model.

One way to avoid the described disadvantages is to take the optimization algorithms out of the translators. Independency and flexibility are achieved, because the optimization methods can be extended easily by the developer. Another way is to undertake aggressive optimization strategies. These strategies could not be valid in the common case, because they are specific for the domain or the concrete application.

Software applications can be treated as models, from which optimal-working systems are obtained. The source code remains unchanged, because the subsystem takes care of the system optimality. One of the important advantages of the approach is that conceptually different software transformations depending on the objectives can be applied. Besides, the techniques serving to achieve its optimal execution could be taken out of the basic algorithm.

The presence of a specialized software subsystem allows division between the model produced by the developer, and the model (for execution) generated by the translator. Consequently, to obtain optimal applications it is not necessary to change the model for the developer.

During translation to an executable model multiple representations (models) are used. The transition from one model to another is not a novelty in software engineering. For the time being the models are substantially different, i.e. there is a big difference among their levels of abstraction. Because of this, many of the transformations from one model to another remain unclear and often irreversible.

Using multiple models with clear and observable mechanisms for transformation allows more accurate translation (transformation) and optimization of the computer programs. This idea is fundamental for the developed framework – SolidOpt.

The design of such a multi-model architecture (shown in Figure 1) generalizes the transformation of the high-level software model into a model, executable by VES. Depending on the goals the transformation can traverse at different stages.

In the framework SolidOpt a transformation optimization method (shortly called optimization method or optimization) is a module, which transforms the software program. This transformation should satisfy given conditions. Most often these conditions are some kind of a metric, which estimates the system quality. Such metrics are the metric for efficiency, the metric for energy consuming, and so on. The different optimization methods use models of different levels of abstraction because of:

- The more efficient method operation – there are situations when an optimization can be applied at several levels of abstraction. However, at different levels the optimization efficiency can be different. Therefore, it is better to choose the model upon which the optimization has produced the best results;
- The easier implementation of the method – for example finding dead code is much easier in an abstract syntax tree (AST) model (the problem is reduced to graph connectivity) than at source code level.

There are methods, which need a concrete model. For example, the machine-dependable optimizations are made at a low level of abstraction (execution level). The implementation of a method replacing the multiplication by a power of 2 with left shift [25] illustrates the machine-dependable optimizations.

One of the objectives of SolidOpt is to give enough models for applying optimizing transformations.

The following scheme, which describes the multiple model interaction of the framework, is valid:

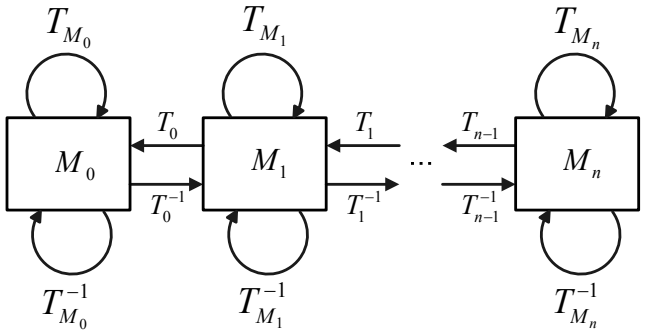

Figure 1. Multiple Model Interaction

Figure 1 uses the symbols:

- M_i – level of model abstraction, where $i \in [0;n]$; M_0 is the objective model, i.e. M_0 is the executable program;
- $T_{M_i}, T_{M_i}^{-1}$ – model transformation, where $i \in [0;n]$. The level of abstraction is preserved;
- T_i, T_i^{-1} – trans-model transformation, where $i \in [0;n-1]$. The level of abstraction is changed.

The scheme describes the relationship between models and ways to lower or raise the level of abstraction. It also shows the mechanism for transforming models. It should be mentioned that there might be other models at the same level of abstraction, and they are called models with equivalent level of abstraction. For example, assume that the model M_0 is designed for CISC processor. It is clear there exists a model M_0', that is designed for RISC processor. We say that M_0 and M_0' are with equivalent level of abstraction.

Forward direction (lower abstraction) is known as compilation. We denote it by $T_i, i \in [0;n-1]$. The increasing level of abstraction (opposite direction) is known as decompilation. We denote it by $T_i^{-1}, i \in [0;n-1]$. An important part of the discussion is the existence and acceptability of these transformations.

When we talk about the transformation of a model in another model, correct transformation functions do not always exist. Sometimes it is acceptable to use an approximation of the function, which transforms one model into another. Such an example is the compilation. The generation of unoptimal code by the translators is partly a consequence of this "satisfying" approximation. The approximation is satisfying, but only in terms of obtaining semantically equivalent models. However, that satisfying approximation may be unsatisfying in terms of optimal execution of the compiled system. Models in SolidOpt model hierarchy are selected to have adequate transformation functions. Therefore, one of the main objectives of the framework is to change the abstraction level of models without the loss of information. When there is no way to avoid it, the loss should be minimal.

Let us consider the transformations $T_{M_i}, i \in [0;n]$ and $T_{M_i}^{-1}, i \in [0;n]$. In essence, they represent the transformation of the model into a model with equivalent level of abstraction (most often in the same type of model). $T_{M_i}^{-1}$ aims to reverse the transformation made by T_{M_i}. It should be noted that conditions for existence are not necessary. In other words, if there is such a function, it could be used for easier manipulation with the model.

Important for the conceptual action of the framework are the functions T_{M_i}. They are used to convert the model into another, which is better from some point of view. The idea can be described by using first order logic. We can define a predicate, which indicates whether the new model is a better one. Define the predicate $P(m,m') = \mu(m') < \mu(m)$, where $\mu(x)$ is a function determining the size of the program model $x \in M_i, i \in [0;n]$. Each transformation, which leads to the satisfaction of P, improves the model in terms of achieving a minimum amount of programming code.

Let us consider the following scheme:

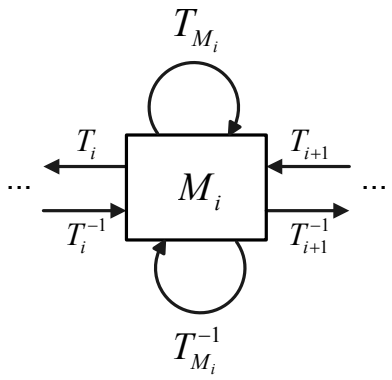

Figure 2. Refactoring / Obfuscation

The scheme represents the i[th] ($i \in [0;n]$) element of Figure 1. Without loss of generality, assume that M_i is the source code of the program. We can define the term refactoring [16] in the following way. Let P be a predicate that satisfies the definition of refactoring. If after the application of transformation T_{M_i} the predicate is satisfied, we say that we have refactored the code.

If $T_{M_i}^{-1}$ exists, we could easily reverse the model without an additional overhead into the pre-T_{M_i} position, i.e. the applied transformation is reversible.

In this scheme we can naturally define another well-known technique for transformation of the code – obfuscation [17]. The technique uses code transformation for another purpose – to make difficult the reverse engineering.

The use of the methodology would facilitate the compilation of optimization methods, operating at different levels. This provides flexibility in the development of sophisticated optimizations such as the merging of classes and DB restructuring. To achieve better results some optimizations may need to work on several abstraction levels of the model.

Using multi-model architecture decreases the productivity of the system. It also increases the overall complexity of the framework. Nevertheless, it should be

noted that the purpose of our framework is to provide a mechanism for optimizing software applications and lower productivity is acceptable. Optimizations compensate the lower productivity of the framework, since greater efficiency of the target system is more important than the time spent in the optimization process. The overall productivity depends on the application domain and the particular application.

3 OPERATIONAL ENVIRONMENT

Software systems are not fully autonomous and independent. Usually they depend on many external factors such as the operating system. In operating systems software systems are separated as processes.

Computer programs should be considered as a combination of processes, which are executed at runtime and operational environment. The combination provides more accurate information about the behaviour.

Consider the following conceptual scheme:

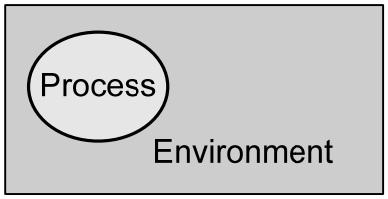

Figure 3. Abstract Operational Environment

It is clearly visible that the processes are placed in the environment and depend on its nature. The abstract scheme helps to obtain general understanding of the following scheme:

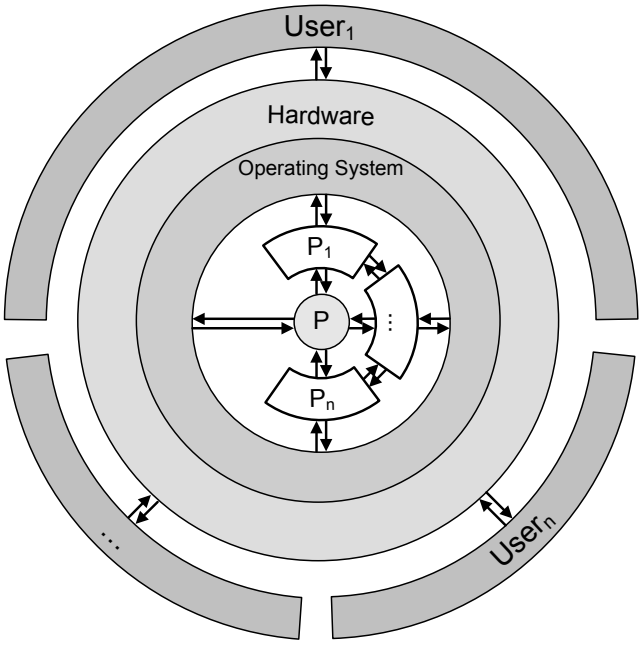

Figure 4. Operational Environment

Environment is composed of three main elements:

- Software that exchanges data with the application – communication between two separate processes is done only through the exchange of data. There are two types of communication – direct and indirect. Direct communication is realized by sending messages (requests) and receiving results. Indirect communication is via a third process (mediator). The type of data and communication determines the great part of the process behaviour;
- Hardware – the characteristics of hardware components define many functional features of the process. For example, a web server process is not fully functional if there is no hardware network card (LAN card). Hardware determines the overall productivity of the process. A special place must be attributed to the VES (or the processor), because it determines the model of lower level of abstraction (M_0), i.e. it has a significant impact on the multi-model architecture;
- The user – a process which exchanges data with the application (it may be a person, but not necessarily). The user defines the so-called user control flow and user data flow of the application. User control flow of the process represents the sequence of operations, which are performed by the user on the process. User data flow is the sequence of data exchanged with the process.

The communication between the user and the hardware on one side, and the software process on the other, is done indirectly – through the hardware and the operating system.

The data, depending on the operational environment, determines the behaviour of the software systems during runtime. The fact is that often the actual behaviour during runtime differs from the behaviour during development. Both behaviours are semantically equivalent, but the differences are mainly in the decreased productivity as a result of different operational environments.

Therefore, it is necessary to analyze the data and to undertake a strategy to adapt the model to it.

For example, let us have the following structure at the source code level:

```
if (x == 1) {y=5}
else if (x == 2) {y=12}
...
else if (x == 29) {y=7}
else if (x == 30) {y=4}
```

We know from the data obtained during runtime that the value that x gets most often is 30. We can apply the transformation of the model and obtain:

```
if (x == 30) {y=4}
else if (x == 1) {y=5}
else if (x == 2) {y=12}
...
else if (x == 29) {y=7}
```

Thus, the program is going to do (most often) 29 comparisons less, which increases its productivity while keeping its semantics. The transformation changes the program control flow based on the user's data flow.

Another illustration of the need to adapt the model according to the operational environment is the architectural changes of the hardware. For example, the design of productive computer programs in the absence of the cache memory (Intel 8086) suggests a specific transformation of the model. Adding cache memory to the architecture of the VES (for instance Intel Pentium) is an architectural change. The transformation executed on the new architecture may not achieve the expected optimal execution. The consideration of the operational environment makes optimizations more efficient.

4 Evolution of Model Optimality

Collecting behaviour data and the idea of improving the system is similar to an evolutionary process. To meet the definition of evolutionary software system, the system should add and/or modify its functionality. Our system for optimizations is based on a similar mechanism, which we call evolution of model optimality. The evolution of model optimality is usually in a specific direction, for example improving some metric such as the size of the program, occupied memory, performance, and power saving. Currently, the primary objective is to improve these metrics without modifying the functionality semantics.

The metric is a quality evaluation function of the model transformation. In other words, it specifies how close the new model is to the desired goal. An interesting discussion, which will be the subject of future investigations, is the combination of several metrics, i.e. the optimization according to several criteria simultaneously. For instance, this could be achieved by determining the weight of every transformation – 20% improved performance, 30% improved application size and 50% improved power saving.

After we have discussed the need of behaviour data (including operational environment data), we have to mention how it will be used. The proper interpretation of the information is more important than the information itself, since incorrectly interpreted reliable information about the system behaviour can cause much greater trouble than the opposite (if we interpret the information correctly we can determine if the information is incorrect). We will divide the methods for analysis according to their complexity. These methods provide guidance on the mechanism, which will be used to interpret the data:

- Simple mathematical models and heuristic methods – we can try to find a relation in the data and a formula that most accurately describes them (for example the Least Square Fitting method or other numerical methods). This provides a relatively simple but powerful mechanism for data interpretation and management. Often, finding such a function is not an easy task. If the specificity of the data is changed, it is necessary to perform the process modelling and function searching once again. Using mathematical models and heuristics strongly depends on the problem area. For example, it is easy to find a heuristic for the application size, while it is hard to find such a heuristic for the system performance or for system power saving (for further information see [18]). The human factor is very important in the process;
- Statistical methods – the use of more sophisticated mathematical models for data analysis is not a novelty. There are many tools and libraries that make quite good data analysis (for further information see [19]). Conclusions are based on the statistical approaches. They can be used to produce better heuristic functions;
- Expert systems – can be used in several directions to achieve flexibility in the conclusions. They can control the optimizing modules, based on the knowledge of experts, and are able to handle large amounts of data automatically. They can be used in combination with statistical methods to ensure precise control over the optimization of computer programs;
- Intelligent Systems – the next step is the use of fully automated systems that control the entire process of optimization. The application of agents, which assist developers in the refactoring such as [20] with the aim to optimize the program models, is a good approach to use and interpret data. It has its advantages and disadvantages, which will not be discussed in this paper.

Our research is primarily oriented towards using statistical methods and expert systems in SolidOpt. Best results would be provided by their combination.

The above can be summarized by the following architectural pattern:

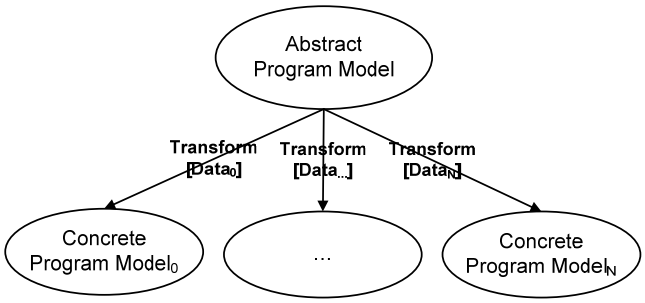

Figure 5. General Program Specialization

Once having the program model and the information about its behaviour during the execution, the evolution of model optimality can begin. On the basis of the obtained information from the interaction with the operational environment and the internal behaviour of the system, an optimization strategy for the model is built. From the program model many concrete models that are optimized according to the received information can be obtained. It should be noted that this is very important when the system has many users. This way, an optimized copy for each individual user is obtained.

Let us consider in detail the following relation: abstract program model (APM) – transformation – concrete program model (CPM).

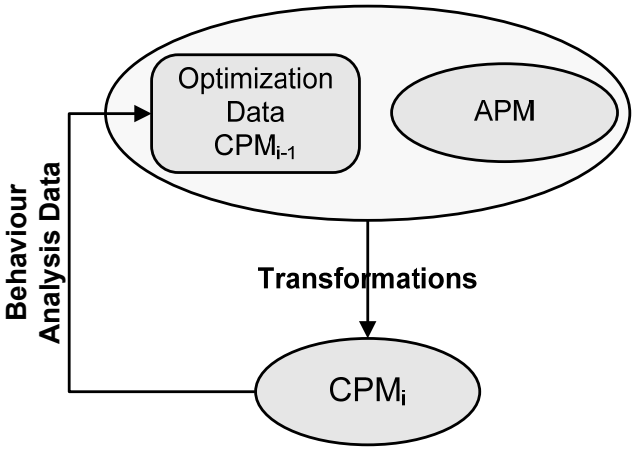

Figure 6. Program Transformation Evolution

The evolutionary pattern describes an endless optimization cycle. From the initial model by transformations we construct a concrete program model, which is VES executable. The execution collects new behaviour data for the next cycle of evolution. For the generation of a new program model one can use optionally:

- The old model (CPM_{i-1}) – reduces the time to adapt (achieving system optimal operation);
- The abstract/initial model (APM) – provides a reliable adaptation of the model to the data. Moreover, sometimes evolution from one specific model to another is impossible;
- The data collected from the operational environment – ensure accurate use of optimizations based on the behaviour of a particular model in the environment.

Attention should be paid to the fact that any evolution period strictly depends on the specifics of the system. It also depends on the time necessary to collect sufficient information for interaction with the operational environment. Not all evolutionary periods are at equal intervals of time.

One important characteristic of the systems in view of the facts discussed above is the adaptability of the model. Model adaptability is the ability of the software system to be a subject of the evolutionary process. Highly specialized applications are more difficult to adapt due to the specific knowledge that is used during development.

When discussing evolution of the model, we need to mention the relation with time. Let us assume that the adaptability of the software is proven. If the time for adaptation is 100 years, for example, it would be irrational to think that it is useful. Therefore, an important property of the adaptability is its acceptability.

Adaptability and its properties are factors that significantly influence the performance of the developed framework SolidOpt. So it will be a subject of future detailed studies.

5 RESULTS

At the moment an experimental framework SolidOpt is being developed based on the following principles:

- *Generality* – the underlying concepts and principles should be applicable to a wide range of application domains, each associated with appropriate software architectures;
- *Openness* – the possibility of a system extension in one or more directions;
- *Extensibility* – the possibility of attaching new functionality by modules;
- *Flexibility* – the easy adaptation to the different domains of applications;
- *Loose coupling and modularity* – the loose coupling increases the possibility of some class being used separately and allows the given system to be easily examined, changed and extended. The modularity is an implementation aspect of the loose coupling.

As a result of the development concept of SolidOpt, preset templates for optimizations can be prepared. They set initial points that reduce the adaptation time.

We work on a flexible configuration mechanism, witch is a realization of the ideas described in [23]. The main idea is for the mechanism to be implemented as an optimizing module. Table 1 shows the results of the preliminary performance tests of our system. We have also provided a comparison with similar systems using different programming languages, frameworks, and file formats.

The increase in the productivity is obvious. Integrating such a module in SolidOpt would improve the efficiency of the optimized applications.

6 APPLICATIONS IN EDUCATION

SolidOpt framework is used in the training of students studying Informatics at the University of Plovdiv "Paisii Hilendarski". It provides a good basis for experimental work of the more advanced students, who want to become familiar not only with the development of translators but also with the use and development of optimization methods. This stimulates the creativity and the innovative thinking of the students.

Another application of SolidOpt is as a basis for the development of graduation theses in software optimization.

Tested on Intel Core 2 Duo T5750 @ 2GHz, 2x2GB RAM Table 1

Programming Language	Test Type	Framework	Param. Count	Parameters Type	Iterations Count	Elapsed Time (ms)	Time/Call AVG (ms)
C#	INI – SolidOpt	.NET 2.0	1	Integer	10^7	63	0,0000063
C#	INI – Native Call	.NET 2.0	1	Integer	10^5	15319	0,1531900
C#	XML – .NET	.NET 2.0	1	Integer	10^7	10686	0,0010686
JAVA	INI – ini4j	JAVA SE 6	1	Integer	10^7	3482	0,0003482
Delphi	INI – VCL	VCL	1	Integer	10^5	14571	0,1457100
C#	INI – SolidOpt	.NET 2.0	1	String	10^5	15	0,0001500
C#	INI – Native Call	.NET 2.0	1	String	10^5	16536	0,1653600
C#	XML – .NET	.NET 2.0	1	String	10^5	156	0,0015600
JAVA	INI – ini4j	JAVA SE 6	1	String	10^5	46	0,0004600
Delphi	INI – VCL	VCL	1	String	10^5	14508	0,1450800
C#	INI – SolidOpt	.NET 2.0	2	Integer, String	10^5	31	0,0001550
C#	INI – Native Call	.NET 2.0	2	Integer, String	10^5	31575	0,1578750
C#	XML – .NET	.NET 2.0	2	Integer, String	10^5	312	0,0015600
JAVA	INI – ini4j	JAVA SE 6	2	Integer, String	10^5	118	0,0005900
Delphi	INI – VCL	VCL	2	Integer, String	10^5	48235	0,2411750
C#	INI – SolidOpt	.NET 2.0	20	String	10^5	16	0,0000080
C#	INI – Native Call	.NET 2.0	20	String	10^3	3120	0,1560000
C#	XML – .NET	.NET 2.0	20	String	10^5	30920	0,0154600
JAVA	INI – ini4j	JAVA SE 6	20	String	10^5	582	0,0002910
Delphi	INI – VCL	VCL	20	String	10^3	2933	0,1466500
C#	INI – SolidOpt	.NET 2.0	100	Integer, String	10^5	31	0,0000031
C#	INI – Native Call	.NET 2.0	100	Integer, String	10^3	16286	0,1628600
C#	XML – .NET	.NET 2.0	100	Integer, String	10^5	12745	0,0012745
JAVA	INI – ini4j	JAVA SE 6	100	Integer, String	10^5	2684	0,0002684
Delphi	INI – VCL	VCL	100	Integer, String	10^3	14851	0,1485100

SolidOpt also provides optimizations, which are applicable to the open system for hybrid geometric modelling OpenF [21]. Experimental optimizations used in OpenF are mostly domain-specific.

An interesting aspect in the student training would be the integration of optimization methods at source code level in e-learning systems such as DeLC [22]. The possibility of integration with refactoring agents of the type described in [20] is an illustration of SolidOpt's interaction with intelligent systems.

7 Conclusion

It was shown that the proposed scheme could be used in practice. From the experiments it becomes clear that there is a necessity of developing the matter. The studied aspects of the framework should be specified in more detail.

An interesting tendency for development is the design of automated methods for optimization based on statistical evaluation of the system operability (profiling).

Of particular interest are the functions $T_{M_i}, T_{M_i}^{-1}$ described in Section 2, since they affect the overall operation of the framework and the optimized applications.

It is interesting to note that in our approach the framework can be controlled by meta-information. Meta-information should be in the form of attributes that allow the optimizing part to be separated from the basic logic of the applications.

SolidOpt will facilitate the development of optimal-working applications, and will provide a mechanism for optimizing applications throughout their entire lifecycle.

References

[1] FxCop, http://msdn.microsoft.com/en-us/library/bb429476.aspx, (visited in March 2009)

[2] StyleCop, http://code.msdn.microsoft.com/sourceanalysis, (visited in March 2009)

[3] Gendarme, http://www.mono-project.com/Gendarme, (visited in March 2009)

[4] Hovemeyer D., Pugh. W., “Finding Bugs Is Easy”, http://www.cs.umd.edu/~pugh/java/bugs/docs/findbugsPaper.pdf, 2003
[5] PMD/Java, http://pmd.sourceforge.net, (visited in March 2009)
[6] Corbett J., Dwyer M., Hatcliff J., Laubach S., Pasareanu C., Robby, Zheng H., “Bandera: Extracting Finite-state Models from Java Source Code”, In *Proceedings of the 22nd International Conference on Software Engineering*, pages 439–448, Limerick, Ireland, June 2000
[7] Artho C., “Finding faults in multi-threaded programs”, Master’s thesis, Institute of Computer Systems, Federal Institute of Technology, Zurich/Austin, 2001
[8] Jlint, http://artho.com/jlint, (visited in March 2009)
[9] Flanagan C., Leino K., Lillibridge M., Nelson G., Saxe J., Stata R., “Extended Static Checking for Java”, In *Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 234–245, Berlin, Germany, June 2002
[10] Rutar N., Almazan C., Foster J., “A Comparison of Bug Finding Tools for Java”, In *Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering*, France, November 2004
[11] Srivastava A., Eustace A., “ATOM: A System for Building Customized Program Analysis Tools”, In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, pages 196–205, Orlando, FL, June 1994
[12] Srivastava A., Edwards A., Vo H., “Vulcan: Binary Transformation in a Distributed Environment”, Microsoft Research Technical Report, MSR-TR-2001-50
[13] Tilevich E., Smaragdakis Y., “Binary refactoring: Improving code behind the scenes”, In *Proceedings of International Conference on Software Engineering (ICSE)*, pages 264–273, May 2005
[14] Vall´ee-Rai R., Hendren L., Sundaresan V., Lam P., Gagnon E., Co P., “SOOT – a Java Optimization Framework”, In *CASCON 1999*, pages 125–135, 1999
[15] Phoenix Compiler and Shared Source Common Language Infrastructure, http://research.microsoft.com/Phoenix/, (visited in June 2008)
[16] Fowler M., “Refactoring: Improving the Design of Existing Programs”, Addison–Wesley, 1999
[17] Low D., “Protecting Java Code via Code Obfuscation”, *ACM Crossroads* 4, No. 3, pages 21–23, 1998
[18] Varde A., Ma S., Maniruzzaman M., Brown D., Rundensteiner E., Sissonjr R., “Comparing mathematical and heuristic approaches for scientific data analysis”, *ACM Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* Vol. 22, Issue 1, January 2008
[19] Law A., “Statistical analysis of simulation output-data: The practical state of the art”, In *Proceedings of the Winter Simulation Conference*, pages 67–72, 2004
[20] Glushkova T., Stoyanova A., “Interaction and adaptation to the specificity of the subject domains in the system for e-Learning and distance training DeLC”, In *Proceedings of the Informatics in science knowledge Conference*, Varna, June 2008
[21] Penev A., Dimov D., Kralchev D., “Open hybrid system for geometrical modeling”, In *Proceedings of the 17th International conference SAER-2003 Conference, Varna*, vol. 1, pages 131–135, 2003
[22] Stojanov S., Ganchev I., Popchev I., O'Droma M., Venkov R., “DeLC – Distributed eLearning Center”, In *Proceedings of the 1st Balkan Conference in Informatics BCI'2003*, Thessaloniki, Greece, pages 327-336, November 2003, ISBN 960-287-045-1
[23] Penev A., Dimov D., Kralchev D., “Acceleration of structured and heterogeneous configuration of the applications”, In *Proceedings 36th conference – Union of Bulgarian Mathematicians*, Sofia, pages 327–331, 2007
[24] Aho A., Lam M., Sethi R., Ullman J., “Compilers: Principles, Techniques, and Tools (2nd Edition)”, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, 2006
[25] Bradley D., “Assembly language programming for the IBM Personal Computer” (in Bulgarian), pages 82–85, 1984