# SYRCoSE 2009

Editors:

Alexander Kamkin, Alexander Petrenko,
Andrey Terekhov

Proceedings of the Third Spring Young Researchers' Colloquium on Software Engineering

Moscow, May 28-29, 2009

Moscow
2009

**Proceedings of the Third Spring Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2009)**. May 28-29, 2009. – Moscow, Russia:

This issue contains papers presented at the Third Spring Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2009) held in Moscow, Russia during May 28-29, 2009. The selection was based on peer reviewing by program committee. Both regular and research-in-progress papers were considered acceptable for this colloquium.

The topics of the colloquium include compatibility and portability of software, graphical modeling, computer networks and telecommunication protocols, functional and performance testing, automata-based programming, and others.

**Труды Третьего весеннего коллоквиума молодых исследователей в области программной инженерии (SYRCoSE 2009)**. 28-29 мая 2009 г. – Москва, Россия:

Сборник содержит статьи, представленные на Третьем коллоквиуме молодых исследователей в области программной инженерии, который проводился в Москве 28-29 мая 2009 г. Отбор статей производился на основе рецензирования материалов членами программного комитета. На коллоквиум допускались как полные статьи, так и краткие сообщения, описывающие текущие исследования.

Программа коллоквиума охватывает следующие темы: совместимость и переносимость ПО, графическое моделирование, компьютерные сети и телекоммуникационные протоколы, функциональное тестирование и тестирование производительности, автоматное программирование и другие.

ISBN 978-5-91474-013-6

# Contents

# Foreword

In this year Spring Young Researchers' Colloquium on Software Engineering (SYRCoSE) observes the third anniversary. Certainly, three years are not too much for a conference, but in spite of the age SYRCoSE is a recognizable young researchers' forum on software engineering. By tradition, colloquium takes two days at the end of spring, when the whether is usually sunny and, what is even more important, students are still in universities, not on summer vacation. During colloquium we try to create a friendly and well-wishing atmosphere that stimulates communication between participants and facilitates making new friends and scientific contacts.

SYRCoSE 2009 is hosted by State University Higher School of Economics (HSE), famous Russian university in the field of economics, business, and software engineering. The event is organized by Institute for System Programming of RAS (ISPRAS) and Saint-Petersburg State University (SPSU) jointly with HSE.

Program Committee has selected 25 papers that cover different topics of software engineering and computer science. Participants of SYRCoSE 2009 represents well-known universities, research institutes and IT companies such as Center "Bioengineering" of RAS, ISPRAS, Moscow Engineering Physics Institute (State University), Moscow State University, RSFLabs, Saint-Petersburg State Polytechnic University, SPSU, Saint-Petersburg State University of Information Technologies, Mechanics and Optics, Tomsk State University, and ZAO "EC-leasing".

We would like to thank all participants of SYRCoSE 2009 and their advisors for very interesting papers. We thank PC members and reviewers for their hard work. In this year we involve our young colleagues for reviewing papers. Thus, colloquium can be called "Young Researchers' and Young Reviewers' Colloquium". Finally, our special thanks to Prof. Sergey Avdoshin, chief of Software Engineering Department of HSE, for his invaluable help in organization of the colloquium.

See you next year at SYRCoSE 2010!

Alexander Kamkin, Alexander Petrenko, Andrey Terekhov
May 2009

# Organization

SYRCoSE 2009 is organized by Institute for System Programming of RAS, Saint-Petersburg State University and State University Higher School of Economics.

## Colloquium Chairs

Alexander K. Petrenko
*Institute for System Programming of RAS, Russia*

Andrey N. Terekhov
*Saint-Petersburg State University, Russia*

## Organizing Committee

Sergey M. Avdoshin
*Higher School of Economics, Russia*

Alexander S. Kamkin
*Institute for System Programming of RAS, Russia*

## Program Committee

Sergey M. Avdoshin
*Higher School of Economics, Russia*

Victor P. Gergel
*Lobachevsky State University of Nizhni Novgorod, Russia*

Vladimir I. Hahanov
*Kharkov National University of Radioelectronics, Ukraine*

Vsevolod P. Kotlyarov
*Saint-Petersburg State Polytechnic University, Russia*

Alexander A. Letichevsky
*Glushkov Institute of Cybernetics, National Academy of Sciences, Ukraine*

Igor V. Mashechkin
*Moscow State University, Russia*

Alexander S. Mikhaylov
*Moscow Engineering Physics Institute (State University), Russia*

Valery A. Nepomniaschy
*Ershov Institute of Informatics Systems, Russia*

Ruslan L. Smelyansky
*Moscow State University, Russia*

Valeriy A. Sokolov
*Yaroslavl Demidov State University, Russia*

Ivan I. Piletski
*Belarusian State University of Informatics and Radioelectronics, Belorussia*

Vladimir V. Voevodin
*Research Computing Center of Moscow State University, Russia*

Nina V. Yevtushenko
*Tomsk State University, Russia*

# Designing a Development Environment to Support Creation of Standard-Compliant Applications

Denis Silakov
Institute for System Programming
at the Russian Academy of Sciences
Moscow, Russian Federation
Email: silakov@ispras.ru

***Abstract*—This paper presents an approach of developing a special environment to help application developers to create programs compliant with some interface standard. The paper suggests a design of the informational system aimed to make it easier to develop and to maintain such an environment on the basis of the existing systems, with necessary modifications in the areas concerned by the standard. To store information about existing systems and to facilitate their modification, it is suggested to use a database with a set of accompanying tools. Necessary aspects of the database schema design are described, as well as some aspects of the tool architecture.**

***Index Terms*—Data management, Reverse engineering, Software requirements and specifications, Software standards.**

## I. Introduction

In the modern software world, it is a common situation when several concurrent implementations exist that provide the same functionality to their users (either human beings or other software applications). From the user's point of view, it would be nice if all systems with the same functionality (that are targeted to solve the same problems) have the same interaction interface – in this case it wouldn't be very hard to replace one implementation with another (that was found to be faster, more convenient, etc.) and users will not be bound to a particular solution. In order to achieve such interchangeability, different interface standards are being developed that specify the set of interfaces that every compliant system should provide.

However, in addition to the standard set of interfaces every system is usually not forbidden to provide some extra ones, that can be unique to it and absent on other platforms. This raises a problem for those developers who want to create applications interacting with the system by means of the standard interfaces only (in order to guarantee that the application will be able to interact with any other system compliant with the same standard). For example, in order to make the application sources be compilable by any C language compiler that is compliant with the ANSI C standard, developers should use only those language constructions and library routines that are defined in that standard; in order to make the application binaries able to be executed on any distribution compliant with the Linux Standard Base [1], they should use only LSB interfaces, and so on. But how to ensure that the application doesn't use any extra interfaces? To be sure, developers can consult documentation on every external interface used by the program. However, this can take significant time, especially if developers are not yet familiar with the standard. Moreover, sometimes dependencies on undesirable interfaces may appear indirectly, as a side effect of the environment used to build the program. For example, for binary executables some dependencies on binary interfaces can be introduced by compiler on the basis of its options, environment variables, etc. Sometimes it can be rather hard to detect and eliminate such indirect usage.

One of the possible ways to automate the control of used interfaces is to create a tool (or use an existing one, if any) that will check used interfaces and report any violation of the standard. If integrated in the build process as a part of the tests, such a tool can be used to detect wrong interfaces at early stages of the development process. However, the checker itself can only state a fact of violation, but not to point out the reason and help to fix it. In particular, it doesn't solve the problem of interfaces that are not used directly but introduced by the build environment.

An ideal solution would be to use a real system that is compliant with the standard and at the same time doesn't provide any non-standard interfaces. Unfortunately, for many standards such implementations don't exist, and it can be quite expensive (or even impossible) to create one from scratch. It appears to be more reasonable to take an existing system compliant with the standard, to cut its non-standard pieces and to set it up properly. Such a restricted environment can be either a self-sufficient system (like the LSB Sample Implementation [2]) or be integrated in the existing system, providing only those components that are covered by the standard (like the OpenGL Sample Implementation [3] or the LSB Development Environment [4]).

One should keep in mind here that every standard evolves over time, and all the tools accompanying the standard should be kept in sync with it. Furthermore, it is not uncommon when several versions of the same standard are demanded by software developers, so specification developers should either support a set of separate versions of helper tools or create tools that can support any specification version from some given set. Modern interface standards often describe hundreds or even thousands of interfaces and evolve quickly by adding more and more interfaces to satisfy market needs. Surely, all accompanying tools should not be left behind; however, the process of their development and support can be even harder then the development of the standard itself. Nowadays, an approach is required to organize, facilitate and automate (where possible) this process to allow specification creators to concentrate on their primary target – specification text.

The remainder of the paper is structured as follows: Section 2 observes some existing approaches of helping application developers to create standard-compliant programs. Section 3 introduces an approach that uses a database with information about standard elements to generate a desired environment in a semi-automatic way and allows to keep the environment in sync with the specification text and other relative tools. Section 4 describes the application of the approach to the LSB Development Environment creation process. Finally, Section 5 summarizes the main ideas.

## II. Existing Approaches

One of the most popular approaches to check that the software product is compliant with some standard is to run appropriate tests (certification tests, if possible) against the final product. For example, if application developers target the Linux Standard Base specification, they can integrate the Linux Application Checker tool [5] to the testing process. When targeting Solaris OS, developers can use the appcert [6] tool to check if their binary files satisfy the rules defined in the Solaris ABI (e.g. don't use private symbols or don't link Solaris libraries statically).

Both advantages and disadvantages of this approach come from the fact that it doesn't affect the development process itself. The main advantage for the developers is that they should not make any modifications to their usual process; they should just add an additional test suite to be executed as a part of the product tests. Surely, this will increase the execution time of the tests, but this is usually not a great issue,

A real disadvantage of the approach is that it only allows to give a verdict on whether the product meets standard requirements. If the tests pass, then everything is alright; but in case of failures, the tests can only postulate the fact that a failure occurred, usually with description of the inconsistency found. It is application developer who should find the reason of the failure in the source code (or build environment) and fix it. Sometimes this may require just a little code modification, but in some cases this may lead to redesign of the product architecture. If no special actions to satisfy the standard were taken before the test execution, then nobody can predict how compliant the result product is and how much efforts will it take to make it fully compliant.

Any preliminary actions taken to make the product more compliant with the standard potentially make this product closer to the specification requirements, thus decreasing the probability of discovering serious inconsistencies when executing the tests. The most evident action is to consult the standard for every interface used by the program to check if it is allowed, which restrictions are put over it, which alternatives can be used, etc. This is probably reasonable in case when developers and architectures are familiar with the standard and can give the answer instantly, without actual addressing of the specification text. Thus, a certain level of expertise is required. Surely, the experience can be obtained through the practice, but for many standards (and for many developers) such straightforward practice as careful investigation of the specification text can significantly delay the release of the product.

A more complicated approach is to provide developers with a special environment that will simplify the process of achieving a standard-compliant product by discovering the inconsistencies as early as possible. For example, the environment can be constructed in such a manner that no program can be built inside it until all standard requirements are met.

The practice of providing such an environment is rather widespread in those areas where direct programming in the target system (where the application should be executed) is difficult – the system can be expensive and difficult of access for developers (e.g. operating systems on mainframes) or have limited resources making it impossible to run such tools as debugger or profiler (e.g. operating systems on mobile devices). In most of such cases, the ideal implementation (that is a system that provides those and only those interfaces that can be used by applications) does exist, but is hard to access. The common solution for this problem is to emulate the target environment on some other system which is more accessible for developers. In order to achieve this, one can either to use a hardware emulator to execute an existing system inside it, or recompile the system for another platform. The former case usually doesn't require any modifications of the system itself, but requires an appropriate hardware emulator. On the opposite, the latter one doesn't require any additional tools but its availability depends on adaptability of the system source code.

An example of the first approach is a QEMU-based Android Emulator [7] that provides a virtual ARM mobile device with a full Android system stack running on it. Android Emulator is an essential part of the Android SDK, which also includes a set of tools to interact with the emulator. An example of the second approach is the LSB Sample Implementation (SI) – a Linux distribution build on the basis of Linux From Scratch (LFS) [8] (until version 4.0) or rPath [9] (since 4.0). Both LFS and rPath allow to obtain a system meeting given requirements. However, none of these technologies is flexible enough to create a system with any given set of functions (moreover, it's not only tools fault – Linux components are sometimes interconnected very closely, and it can be impossible to cut one component without affecting the others). This fact complicates the development process of the LSB SI itself and it is actually not guaranteed that the result system provides no forbidden interfaces.

Another possible way of emulating a desired environment is to provide some mechanism for one of the existing systems to behave exactly as the desired one. Such approach can be used even if the desired environment doesn't exist in a pure form (i.e. if there is no real system that provides only allowed interfaces). However, required modifications of the system can be significant, and in case of complex systems it can be rather difficult to maintain these changes in parallel to the development of the main system. So this way is acceptable when implemented and maintained by developers of the main system, and when required possibility is an integral part of that system. As an example of this approach, we can mention C and C++ compilers from the GNU Compiler Collection that support '-std' option that can be used to determine the language standard to be followed by compiler (such as ISO C90 or ISO C++98).

The approaches mentioned above concern modifications and adaptation of rather complex software systems, such as operating system or compiler. Development of required modifications is usually a separate project, whose complexity depends on different conditions, such as size of required modifications, source code adaptability and so on. Sometimes the required efforts are minimalistic, sometimes they are not. In any case, at the current state of art a specific approach is used for every particular project of such kind, but almost all these projects imply manual examination of the system, detection of pieces that are not compatible with the given requirements and investigation of possible fixes for these pieces. Thus, such approaches are reasonable only if there is already a system that is rather close to the desired one.

However, in many cases there is no need to modify the whole huge software product (e.g. operating system). Instead, it is enough to provide alternative implementation for some of its parts. Note that there is a significant difference between providing an alternative implementation that coexists with the default one (or replaces the default one) and providing the whole system where the desired implementation is the only available one. The latter case requires a separate machine for the system to be executed (either a physical or virtual). In the former case, developers can use their usual systems in their usual work and use alternative environment during application development. In particular, this allows developers to use their habitual IDE, debugger and other tools that are hardly provided by the modified environment.

This way is actually the only reasonable approach for those standards that specify only a small part of the system interfaces. For example, the SGI's sample implementation of the OpenGL API [3] presents several hundreds of OpenGL functions. The main purpose of the OpenGL Sample Implementation is to give a standard base for other vendors of OpenGL implementations (in the first place, developers of implementations that use hardware acceleration), but it can be also used by application developers (since 1999, the Sample Implementation is an open source product) to ensure that they don't use any non-standard GL functions. However, it makes no sense to provide a separate operating system with this implementation, since such a system would have a very small difference with respect to any existing one. Moreover, such a system would not be very convenient for everyday usage, since the OpenGL SI doesn't include any hardware drivers and thus lacks for performance.

A similar way is followed by the LSB Development Environment, also known as the LSB Software Development Kit (LSB SDK), which is discussed in details in the Section 4. This product is designed for application developers and only provides headers, compiler wrapper and stub files for libraries to be used during the build process. The SDK can be used only to compile applications, but not to execute or test them. However, this is enough to guarantee that the application doesn't use non-LSB interfaces (except the cases when indirect calls such as one using the dlopen function are used). LSB SDK proved to be much more easier to be maintained than the LSB Sample Implementation, and on the opposite of the latter it guarantees that its header files and libraries don't provide forbidden interfaces. But since the SDK cannot be used to test applications, it is actually the chain of the SDK and SI that should be used in the development process.

This example is similar to the usage of cross-compilation to create executables for the hardware architectures that differ from the current one – in that case one needs a compiler that supports cross-compilation for the target platform, with appropriate header files and shared libraries. The whole operation system is not required for compilation, though some additional runtime environment is required to execute and test the application.

But even providing of only a subset of the complex system may require significant efforts on synchronizing the environment with the underlying software standard, especially if the standard evolves quickly. A promising approach of facilitating this task was introduced by the original creators of the LSB specification and its accompanying tools, including the LSB SDK – it was suggested to store those properties of the standardized items that are used by the tools in a central database and generate appropriate parts of the tools automatically using this database. Initially the LSB database contained names of standardized interfaces, their signatures and all types required to make these signatures complete. At the very beginning, the database was populated with data manually, but as LSB evolved, some tools were introduced to automate this process.

The rest of this paper summarizes and generalizes the approach used by the LSB team, giving general advises on how to automate the process of environment generation using the database, as well as appropriate data collection and manipulation processes. In addition, some further developments of this approach are given, including support of several versions of the standard by the same set of tools. In the Section 4, we'll return to the LSB SDK and describe its current state, with all improvements made by the authors.

## III. Database-Driven Environment Generation

The approach under discussion suggests to pick out those parts of the tools to be created that are either specified by the standard or influenced by the standardized items. Information about such elements should be separated from data about other parts of the tools, that are not concerned by the standard in any way. That is, one should create a template of every tool – some kind of skeleton that just misses those parts that concern standardized items. These “holes” should be filled by additional *generator* on the basis of some external storage. In this paper, we consider a database to be such a storage, though the main ideas can be applied to any storage kind.

## *A. Database Design*

First of all, one have to detect what should be stored in the database and design appropriate database schema. For every tool to be generated using the database, the following analysis should be performed:

- Determine entities involved in the tool that are defined in the standard.
- Determine entities that are influenced by the standardized items. This step is recursive – one should also determine entities influenced by those that are influenced by the standardize items, and so on, until a closed set is obtained.
- Determine which properties of entities picked out during the first two steps are necessary for the tool.

On the third step, it might be necessary to determine all properties required for the tool, even those that are not concerned by the standard. A clear rule can be used to determine if one should store all properties of some item in the database or only those that are concerned by the standard; to formulate this rule, we should first give a definition of one important term:

*Imitating the relational data model, let’s say that a set of properties of an item forms its* ***primary key****, if in the real world (or in some restricted environment where we operate) this item can be unambiguously identified by this set of properties.*

For example, a binary interface exported by a library written in C language is unambiguously defined inside the library by the name of appropriate source-level item (either function or global variable). That’s why name of the binary symbol of C library is equal to the name of appropriate source item (to be honest, with a small reservation for symbol versions). For C++ language, source-level name is not enough – one should add names of all namespaces and classes to which the item belongs (since functions with the same name can exist in different classes), as well as list of parameters (since every function can be overloaded, i.e. one class can provide several functions with the same name, but with different parameters). That’s why names of C++ binary symbols are constructed in a special way (by means of the process known as *mangling*) to reflect all these traits. In any case, if we consider the whole system, we should add name of the library to the primary key of every interface to unambiguously identify it; if we admit that the system can provide several libraries with the same name in different locations, then we should also add a library location, and so on.

By means of the primary key definition, we suggest to use the following rule to identify the properties that should be stored in the database:

**Advice 1.** *If it is detected that at least part of the primary key of some item should be stored in the database, then all properties of this item (including those that form the primary key) should be stored there. Otherwise, it is enough to store only those properties that depend on the standard.*

This rule is based on the fact that if we can get the primary key of some item without consulting the standard, then we can unambiguously detect the place of this item in the code of the tool to be generated. So we can simply put the item to the appropriate place in the tool template, leaving holes for secondary properties affected by the standard (if any). Since the location of item can be calculated, then it will cause no problems to create a generator that will detect the holes and populate them with data.

In other case, we can’t detect the actual place for the item in the code of the tool. Thus, we have to store all information about this item in some external storage and pick it up when necessary. Moreover, since we don’t know the standardized primary key of the item, we can’t exactly identify the item, and in general we should store information for all primary keys that are acceptable by the standard requirements. Fortunately, in many particular cases these possible primary keys are restricted to a very limited set. For example, as we saw above, in case of functions exported by libraries written in C language, a function name (maybe together with a library name) is enough – that is, there is no need to create several entries in the database for the same function, since all its attributes (parameters, return value) are unambiguously identified by the name.

One may notice that the database itself needs some primary key to store secondary properties of every item. This primary key can be exactly the same as the one of the item, or some artificial key that allows to calculate the ’real’ one (for example, a result of some hash function can be used to achieve a numerical artificial key to decrease the size of the database and to speed up the database queries). So the gain of not storing some properties in the database actually concerns only those properties that are not included in the primary key and don’t depend on the standard. In every particular case, this gain can be estimated; if it doesn’t prove to be significant it may make sense to store all properties of the item in the database.

Thus, if we are not sure if some item is obligatory for the tool and will appear there regardless of other conditions, then we should store all properties of this item in the database. But how generator will know if a given database entry should be picked up during generation? The simplest way is to assign a boolean flag to every database record:

**Advice 2.** *If all properties (including the primary key) of some kind of items are stored in the database, then additional boolean flag should be attached to every entry corresponding to this kind of items to indicate if a particular entry should be picked up by the generator when creating the tool.*

Let’s call this flag as *appearance* flag. With such a flag, generators should only select those entries for which the flag is set.

## B. Handling Item Interdependencies

The most straightforward way to set the appearance flags is to set them manually. However, in some cases this can be a rather complicated task – the thing is that different items can depend on each other, and such interdependencies should be taken into account. For example, if we want to declare some function in a header file, we should also declare all types necessary for its declaration (that is, for parameters and return value) or to include other header files that will provide necessary declarations. The required types can have complex structure and require other types to be declared, and so on. Resolving such dependencies manually is tiresome; however, usually these dependencies can be clearly formulated, and it makes sense to automate the resolving process.

In order to do this, we suggest to create a *markup* tool, that will check and resolve dependencies between different items. In general, such a tool should be provided with a basis – a set of items that should be included in the generated code. This set should be formed manually and should contain all items that are included in the specification. On the basis of this information, the markup tool will decide which additional items should be included to the generated code to make it complete.

For example, let's suppose that the specification defines a set of C functions with signatures, and in the database we store function names, as well as type names, and mapping between function parameters and types, as well as mapping between return values and types. In order to generate fully qualified declaration, developers may only mark necessary functions as included, and the types that should be marked as included will be calculated automatically by the markup tool.

In this example we suppose that types are *subordinate* objects – if a type is required for some functions, then it is included without any discussion. However, another situation is possible, when the items in question are equal in rights – that is, it is not clear, if we should include a type or exclude all interfaces that use it. In case of such situations, the markup tool can only report the problem, but not solve it.

From the generator architecture point of view, there are at least two ways of implementing markup functionality:

- Implement a separate tool that will set flags directly in the database.
- Implement necessary functionality in generators; in this case all dependencies are resolved during generation, and database modifications are not required.

It's hard to say if one of these approaches is better than the other. The disadvantage of the second approach is that it makes the generation logic more complicated, and can slow down the generation process. On the other side, the first approach implies that the markup tool should be executed after any database change that can affect the generated code. For confidence, one can execute this tool every time the generation is launched, so the actual time of generation will also increase. The first approach is more preferable in case when the markup tool cannot resolve dependencies by itself and should interact with the user.

One more important question is how to store information about dependencies (that is, how the markup tool should know that one item depends on another). The straightforward way is to hardcode all the dependencies in the markup tool. However, there is a more elegant solution – if the used DBMS supports *foreign keys*, then it is enough to set such keys for the necessary tables, and the markup tool will be able to use them to detect dependencies. Considering again the example with C functions, one may store functions and their return values in the following way:

- Function names are stored in the *Function* table.
- Type names are stored in the *Type* table; let's suppose that this table has a primary key *Tid*.
- The *Function* table should also have *RetValue* field to store the type of its return value. This field should be a foreign key referencing appropriate *Type* record by its primary key *Tid*.

In general, we believe that this way is much more flexible than the hardcoded dependencies in the tools. So, one more rule:

**Advice 3.** *Interdependencies between different items should be implemented as foreign keys of appropriate tables in the database.*

## C. Populating Database With Data

To design the database schema and to create generators is only a half of the problem. The database is useless until it is populated with data. We have already considered one aspect of the data management by suggesting a tool that will take care of item interdependencies, but this is just an auxiliary tool, a kind of consistency checker, though its usage can save a lot of time. However, the main problem is to collect the 'raw' data – that is, particular interfaces of different kinds with their properties and accompanying items. There can be no general method of solving such a problem, since in different cases the data can have different sources. For example, the whole set of standardized interfaces can be created from scratch by standard developers (e.g. as a result of some scientific research). In this case it is likely that there is no other way of populating database with data except typing the data manually. Some kinds of automation can be available in any case – for example, transforming textual lists of interfaces into SQL statements; however, these textual lists should be also achieved somehow. But here we'd like to point out one important situation when the significant part of necessary data can be obtained automatically. We are talking about the case when the standard is not created from scratch, but is based on some existing system, or generalizes interfaces of several implementations.

**Advice 4.** *In order to populate the database with data, it is useful to create a set of tools that will allow automated extraction of necessary information from existing systems.*

We suggest to extract as much data as possible from existing systems. We believe that there is no need to apply complicated filters to the collected data in order to select only those that have some chances to be useful for the standard. It is much more easier to set/unset the appearance flag for lots of entries than to collect additional data in case if one finds that we haven't collected enough. With the modern database management systems, it is unlikely that specification developers will reach some database limitations on the data size or performance – yes, the modern standards are large, and if we collect and store all the data that can be useful for the developers, we can achieve large amounts of data. For example, the LSB, one of the biggest interface standards in the world, contains about 40,000 interfaces. A usual Linux distribution consisting of one DVD disc can contain up to million interfaces, and theoretically, all these interfaces can be useful for the LSB. However, even these numbers are not a problem for the modern DBMS (in particular, for MySQL, used as the DBMS for the LSB Database which currently contains about 100 millions of records).

The task of data collection is a separate problem and can be even more complicated than generator development. As LSB developers' experience shows, it is much more simple to generate headers with declarations of functions and types on the basis of structured database information, then to parse and analyze existing headers in order to populate the database with this structured data. However, even this task doesn't seem to be very time-consuming when compared to the task of manual collection of data for 40,000 functions and similar quantity of types, constants and macros necessary for their declaration and usage.

Even more gain can be achieved if there are several tools that are generated (at least partially) using the database. Though these tools can require different generators to be created, it is likely that they use similar information from the database. If so, the same tool can be used to populate the database with data satisfying all generators.

### D. Supporting Multiple Versions of the Standard

In the "Database Design" section, we've suggested to use an *appearance* flag to indicate that a particular item is included in the standard. To be sure, a single flag allows to store information corresponding to a single specification version. However, as we have noticed in the beginning, sometimes it is necessary to support several versions simultaneously. Forking a separate copy of the database for every version is a possible approach, but it introduces great data overhead (especially in case when different versions have significant intersection), complicates back porting of fixes for different issues and can significantly increase the maintenance cost of the whole system.

Another possible approach is to use the same database to store information about all specification versions. In this case the same tools can be used to generate data corresponding to a given version (moreover, it is possible to generate a single tool whose behavior can be adjusted by user to correlate with a particular specification version). Detailed description of possible approaches of improving the database to store such data can be found in [10]. Here we'll just summarize the main statements of that work:

**Advice 5.** *In order to store temporal data, one should replace the 'appearance' flag with the time interval that will indicate a set of versions where the item was included in the standard.*

This statement actually suggests to add a temporal dimension to the database. This can be done by using either temporal DBMS, or relational DBMS with additional fields indicating the interval bounds. With respect to specification versions, we may notice the following important features:

- The time is discrete; the possible values of interval bounds are standard versions, broadened with at least one specific value, that can be referenced as 'infinity'. If this value is used as a lower bound of the time interval, then this means that the entry has never been included in the standard. If it is used as an upper bound and the lower bound is not infinity, then the entry is included in all versions of the standard starting with those pointed by the lower bound of the time interval.
- When storing specification versions, one have to deal with only one kind of temporal data, called *valid time*, and don't need to store *transaction time*. That is, we should know in which version of the standard some item appeared, but not the time when the appropriate change was made in the database.

Thus, there is no need to track transaction time, and the set of possible values for the valid time is usually limited to a rather small set of values (fortunately, the standards doesn't introduce a new version every day). Moreover, a common life cycle of an item in the standard looks like *"appeared in version **A**, withdrawn in version **B**"*. In general, situations when some item is returned back after been withdrawn are quite rare. So for most cases the database will contain only one time interval indicated by two bounds – that is, if compared to a single appearance field, we just obtain one more field for every record. Thus, this approach is much more efficient from the data size point of view than a set of separate copies of the database corresponding to particular standard versions.

## IV. Developing the LSB Development Environment

Let's now return to the LSB Development Environment and show how the ideas discussed above are used during its creation.

First, let's note that the main purpose of the Linux Standard Base specification is to pick out those interfaces provided by the operation system that are common to all major Linux distributions. The following kinds of interfaces are taken into account:

- Binary libraries (their runtime names).
- Binary interfaces (functions and global variables) provided by libraries.

- Commands (utilities and shell builtins).
- Modules of interpreted languages.
- Kinds of sections for the ELF files.
- RPM format tags.

That is, the LSB specifies the runtime environment where the application is running, and concerns the problem of distributing the applications in a compiled form. Applications that use only interfaces included in the LSB can be executed on any LSB compliant system without recompilation or environment adjustments.

LSB 4.0 contains specifications for 57 libraries with about 40,000 binary interfaces. These numbers might look large, but not when compared to any desktop Linux distribution – usual system on one DVD disk ships several thousands of libraries and up to million of interfaces. So it's not easy for application developers to orientate themselves in the Linux and LSB world. The developers who want to target LSB are supported with the LSB Navigator – web system that represents the LSB online, with lots of additional helpful information not included in the standard itself – and Linux Application Checker that can be used to check LSB compliance of executable files, shared objects and scripts that form the application.

However, these two tools are not free from the issues discussed in the beginning of this paper – consulting Navigator for every interface requires a lot of time; Application Checker can be integrated in the build process, but this will increase the build time, and in many cases the checker will just report the failure, but provide no suggestions on how to fix it. Application developers can meet some issues with obtaining LSB-compliant product when building their programs in the real systems, even if they don't directly use non-standard interfaces. For example, the following two problems arise quite often:

- LSB doesn't include arithmetic routines of the libgcc_s library. However, the gcc compiler forces usage of these routines if they are provided by the libgcc_s library that participates in the build process.
- Default behavior of the gcc compiler on some systems leads to the usage of ELF sections that are supported only by the last generation of distributions and thus not portable and not yet included to LSB. For example, Avinesh Kumar in his blog [11] explains why binaries compiled on RHEL 5 with the default compiler options will fail to run on RHEL 4.

Without a good knowledge of compiler operational principles, it's not easy to find out the roots of such issues.

To save application developers from these problems, it was decided to support them with the specific build environment, whose usage in the build process will guarantee that the obtained executables are compliant with LSB. At the moment two projects exist that provide such possibility:

- LSB Sample Implementation – the whole distribution built using the rPath technology that tries to limit provided interfaces to LSB ones.
- LSB Software Development Kit – a set of tools that can be installed in any real system and used as alternatives to the system build toolchain.

LSB Sample Implementation (LSB SI) is useful not only for building applications, but also for testing them, since it is guaranteed that implementations of its interfaces are compliant with LSB. However, since LSB SI is a distribution, it requires a separate machine to run; though it can be also used as a chroot environment, but even this variant is not very convenient for many developers. Another problem was already mentioned in the Section 2 – the development process of the SI is rather complicated and it is not guaranteed that it provides no forbidden interfaces at all.

On the opposite, LSB SDK doesn't suffer from these issues. To understand the reason, let's first consider the structure of this environment. It consists of the three major components:

- Stubs for libraries specified by LSB. These stubs export only those symbols that are included in the standard, but they don't provide their implementation.
- Header files that provide API (function declarations, types, constants, etc.) for the LSB libraries. It is guaranteed that usage of this API cannot lead to incompliant application.
- Compiler wrapper – a tool that should be called instead of system compiler. This tool calls the system compiler itself, forcing it to use libraries and header files provided by the LSB SDK. The environment variables and compiler options are automatically set to eliminate possibility of obtaining incompliant applications.

The advantage of the LSB development process, from our approach point of view, is the existence of *specification database* that stores different information about elements included in the specification. In particular, the database stores names of included libraries, as well as names and signatures of included interfaces. This database is used to generate LSB specification text and some primitive tests, and also serves as a knowledge base for LSB Navigator and Linux Application Checker. The data collection process that populates the database with information is separated from all other tasks and can be modified without affecting any other items of the LSB infrastructure. Thus, no wonder that the idea raised to use the database to automate the process of the LSB SDK development by introducing an automatic generation of some parts of the SDK, especially stub libraries and header files, that hardly depend on the entities included in the standard.

Stub library generation process has been implemented in a rather straightforward way. In order to obtain a desired stub, a source file in C language is generated with declarations of functions whose names are the names of binary symbols to be exported. Return types and parameters don't matter anything – we may use **void** as a return type for all functions and totally omit parameters. After this file is compiled, we'll obtain a shared library that exports exactly those symbols that are required by LSB.

This approach uses the fact that for libraries written in C, names of exported binary symbols (that are taken into account by the dynamic linker) are equal to the names of functions and global variables implemented in the source files; return types and parameters are not taken into account. For C++ and other languages that allow users to override functions, the situation is different – names of binary symbols are constructed during the *mangling* process on the basis of the function name itself, its parameters and name of the class or namespace which it belongs to. However, if we know binary symbol name of a C++ function, we may create a C source file with declaration of a function with this binary name and compile it. The result will be the same as if we create a C++ source file and place a proper function declaration there (i.e. with all necessary classes and parameters).

Thus, all that we should know to generate stub libraries are binary names of symbols that should be exported by them. The database fairly provides us with this information. Since the generated source files don't contain any actual code, then there are no interdependencies among functions that should be taken into account.

Unlike library stubs, header files should provide complete declarations of functions. All non-intrinsic types used in these declarations should also been declared. In addition, it is useful to declare such elements as constants and macros. Though are they are out of LSB scope, they can be useful for developers (but one should ensure that the macros don't invoke forbidden interfaces). Thus, though we still don't need function implementations, we have to declare complex types that sometimes have rather tricky interconnections, so generation of correct header files is a more complicated task then generation of library stubs. In case of C++, we also have to deal with templates that can affect runtime dependencies of applications. Every template should be analyzed to decide whether its usage can lead to calls to forbidden functions. This task was found to be rather complicated, and it is not completely solved at the moment, i.e. it is not guaranteed that C++ header files shipped with the LSB SDK cannot lead to usage of non-LSB functions. However, even if the latter happens and application obtains a dependency on a forbidden interface, this will be caught at the linking stage which will fail, since stub libraries don't provide the forbidden interfaces.

In the very beginning, the LSB database was populated manually; at that time, no one thought about SDK generation, and for the specification and test generation purposes the data was complete enough. As LSB evolved, the size of necessary data became too large to be handled manually, and additional tools were developed to automate data collection process (import* scripts, libtodb tool [12]). These tools were mainly based on the analysis of binary library files with debugging information, with slight header files analysis. However, though debugging information from binary files could give enough data to generate specification text and some tests, a lot of manual adjustments were required to make generation of header files possible. To improve data collection process, a new set of tools called LibToDB2 were developed at ISP RAS under the contract with the Linux Foundation [13]. The new tools analyze both binary files and headers, extracting data more accurately then their predecessors. The tools collect and upload to the database all information about analyzed headers and libraries. Then on the basis of lists of functions and global variables included in LSB, a separate tool discovers the types that should be present in header files to declare included interfaces. Surely, LSB workgroup can point out that some additional types are useful for developers and should be included in headers; these types should be marked as included manually. The same situation is with macros and constants, that are mainly selected manually, though there is a tool to check that included macros don't require non-LSB interfaces.

One more challenge was to make the LSB SDK to be a multiversion tool, that is, able to generate code complaint with any given LSB version. The LSB database contains data about all released versions of LSB (as well as about the one under development), so it can give us all necessary information about LSB history [10]. Thus, the task was to generate headers and stub libraries that could be used to target any LSB version. Depending on some environment conditions, the SDK should behave as if it represents a particular LSB version. The actual tool that user deals with is a compiler wrapper; to target a particular LSB version, user should only specify an option for it or to set appropriate environment variable, and the wrapper should perform all other necessary actions.

Since the set of symbols exported by a particular library cannot be affected by its environment, we can't generate a single stub file for every library to target several LSB versions. Instead, we create a separate file for every LSB version where the library appears, and let the compiler wrapper to choose the file to link against at runtime, on the basis of its options and environment variables. Unlike binary libraries, the contents exported by header files can be manipulated using different preprocessor directives. In the LSB SDK, every declaration is embraced with conditions on the LSB_VERSION constant, which can be set either directly by user or by the compiler wrapper. This results in smaller total size of files with respect to the case when a separate header is generated for every version, since we can create only one declaration of element to target several versions, and most elements are present in more than one version of LSB.

To estimate the gain achieved by usage of generators instead of manual creation of the whole SDK, one can compare the sizes of generated files with sizes of those part of the SDK that are written manually and with the size of generators themselves. Table I contains characteristics of the generated C-language part of the SDK (as was mentioned above, for C++ elements only stub libraries are generated at the moment, and generators of header files are under development).

Table II contains the same characteristics of the generators and data collection tools, which are written in Perl.

Table III contains the characteristics of the LibToDB2 set of tools (those part that collect C-language data), which is also important for the SDK generation. Note, however, that the data collected by these tools is used by many generators, not only by the SDK ones.

TABLE I
LSB SDK GENERATED CODE CHARACTERISTICS

| | Stub libraries | Headers |
|---|---|---|
| Source code size, loc | 87,800 | 68,500 |
| Development effort estimate, person-years | 22 | 17 |
| Total estimated cost to develop, dollars | 2,950,000 | 2,300,000 |

TABLE II
LSB SDK GENERATORS CODE CHARACTERISTICS

| | Stub libraries | Headers |
|---|---|---|
| Source code size, loc | 400 | 2,100 |
| Development effort estimate, person-years | 0,07 (1 month) | 0,5 (6 months) |
| Total estimated cost to develop, dollars | 10,000 | 60,000 |

TABLE III
LIBTODB2 CODE CHARACTERISTICS

| | |
|---|---|
| Source code size, loc | 4,400 |
| Development effort estimate, person-years | 1 |
| Total estimated cost to develop, dollars | 126,000 |

The data is generated using David A. Wheeler's 'SLOC-Count' [14] tool. Estimates are given by this tool using the basic COCOMO model. Surely, the estimates suppose that the code is created from scratch and don't take into account the nature of the code (such as absence of function implementations in stub libraries). One may also note that the SDK code can be created on the basis of appropriate upstream code by dropping those parts that are incompliant with LSB. However, such an analysis is rather difficult, and in any case it's not simple to support more than hundred of thousands of lines of code manually.

Schema of the database tables that are used during the LSB SDK generation is shown at Fig.1. Temporal data is stored in the tables that implement many-to-many relationships between entities and hardware architectures, since appearance of an entity on some architecture is independent from its presence on other platforms, as described in [10]. The only exception is header files – if a header is included on one platform, it is automatically treated as included on all others (even if it is empty there). So there is no direct relationship between headers and architectures, and temporal data is stored in the Header table itself.

Starting with LSB 4.0, the LSB SDK can be used to create executables and shared libraries compliant with any given LSB version, greater or equal to 3.0 (earlier versions are not supported by the database, since there was no great demand for them). At the moment, the SDK allows to generate only files fully compliant with LSB, with no exceptions (that is, one cannot use libraries or functions that are not included in LSB). In future it is planned to implement a *relax* mode for the SDK that will allow applications to use certain symbols not included in LSB, but still known to be stable and present on most systems. Such possibility should increase the portability of applications (though formally the programs compiled in the relax mode will not be compliant with LSB). But even at the current state of art the SDK is very useful, and is used to build all programs involved in the LSB infrastructure (including test suites, LSB Application Battery and the SDK itself). Another good example of the SDK usage is the OpenPrinting project that suggest to write all printer drivers using the LSB DDK (Driver Development Kit) which is actually an enhanced version of the LSB SDK (supplemented by tools and libraries necessary for driver development but not included in LSB – such as Ghostscript or CUPS DDK) [15]. All current distribution-independent driver packages in the OpenPrinting database are based on LSB 3.2.

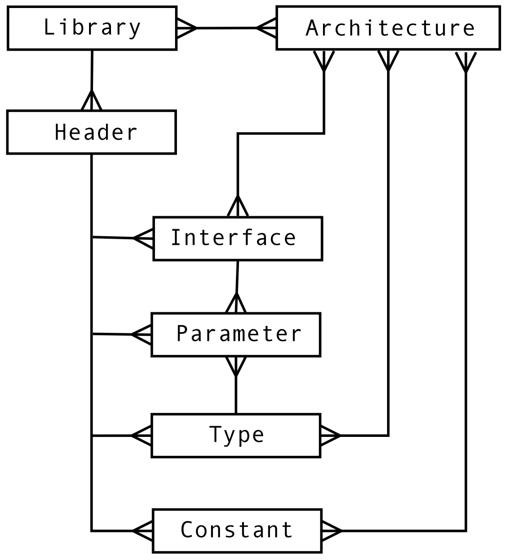


Fig. 1. Main Tables of the LSB Database Used During LSB SDK Generation

## V. CONCLUSION

When creating a portable application, developers are interested in using only those interfaces that are present on the most of the target systems. One of the way of achieving this is to use interface standards that are followed by the target systems. However, direct consulting with the standard text is not very efficient, and consulting appropriate specialists can be expensive; in both cases, the development period can increase significantly. That's why it is important to support developers with toolchains whose usage will guarantee that the final product is compliant with the standard. However, integration of new tools can be also expensive and requires some time. So it is important to provide software developers with the tools similar to those that are already used in their development process, to make it cheap to replace existing tools with the new ones (or use the two toolchains in parallel). One of the possible ways of creating such tools is to modify the existing ones.

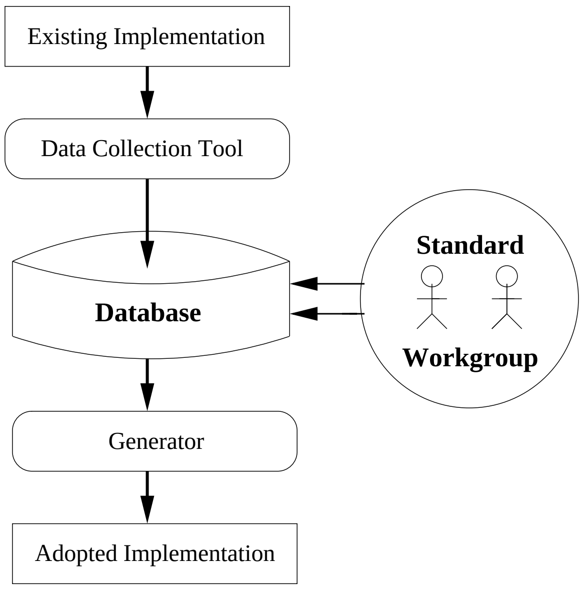


Fig. 2. Developing Adopted Tool Using Existing Implementation

This paper discusses an approach that allows to automate development and support of the tools based on the existing implementations. Fig.2 illustrates the suggested organization of the development process. The main idea is to collect data about existing implementation and to put it to the database in some intermediate format convenient for the specification developers. The data can be manipulated by developers in many ways; in particular, they can use special markers to separate those data that should appear in the tool from undesired or useless information. The automatic generators will use the adjusted data to create the tool itself; the tool achieved is actually a modification of existing implementation, adopted for the specification requirements.

Usage of the database allows specification developers to deal only with those properties of the tool that are concerned by the standard. All other aspects necessary for achieving a working program are handled by automatic generators that also propagate every change made in the database to all places where it should take effect. Thus, the specification developers can even have slight knowledge about the generated tool structure – they just should point out which interfaces are allowed, and which are not. Surely, complex interface standards require creation of complex developer-oriented tools, and the more complicated is a tool, the more complicated are appropriate generators and data collection tools. Nevertheless, development of complicated generators can be much more cheaper than development of the tools from scratch. And even more gain can be achieved if the same database is used to support several versions of tools corresponding to different versions of standards. Adding temporal dimension to the database and support of this dimension by the generators is not as expensive as support of several distinct tools or several distinct databases.

The approach suggested in this paper is used to develop the LSB SDK and proved to be useful and efficient, saving lots of efforts and resources. Since the SDK functions in a similar way as the system compiler toolchain does, its usage is transparent for developers and it is now adopted by many people who care about building software products portable among different Linux distributions in the form of binary files.

## References

[1] Linux Standard Base. http://www.linuxfoundation.org/en/LSB
[2] Building Applications with the Linux Standard Base: Using the LSB Sample Implementation. http://www.linuxfoundation.org/en/Book/HowToSI
[3] SGI's Sample Implementation of the OpenGL API. http://oss.sgi.com/projects/ogl-sample/
[4] Building Applications with the Linux Standard Base: Using the LSB Development Environment. http://www.linuxfoundation.org/en/Book/HowToDevel
[5] All About the Linux Application Checker. http://ldn.linuxfoundation.org/lsb/all-about-linux-application-checker
[6] Becky S. Chu. appcert: A Static Application Checking Tool.Sun Developer Network, June 2001. http://developers.sun.com/solaris/articles/appcert.html
[7] Android Emulator. http://code.google.com/android/reference/emulator.html
[8] Linux From Scratch. http://www.linuxfromscratch.org/
[9] rPath. http://www.rpath.com
[10] Denis Silakov. Tracking Specification Requirements Evolution: Database Approach. Proceedings of the First Spring Young Researchers' Colloquium on Software Engineering (SYRCoSE'2007), Volume 2, pp. 15-22. May 31 - June 1, 2007. - Moscow, Russia.
[11] Avinesh Kumar. Binary incompatibility between RHEL4 and RHEL5. http://avinesh.googlepages.com/binaryincompatibilitybetweenrhel4andrhel
[12] LSB DB Tools. http://ispras.linuxfoundation.org/index.php/LSB_DB_Tools
[13] LSB Library Import Tools. http://ispras.linuxfoundation.org/index.php/LSB_Library_Import_Tools
[14] SLOCCount Tool. http://www.dwheeler.com/sloccount/
[15] OpenPrinting: Writing And Packaging Printer Drivers. http://www.linuxfoundation.org/en/OpenPrinting/WritingAndPackagingPrinterDrivers

# Binary Compatibility of Shared Libraries Implemented in C++ on GNU/Linux Systems.

Pavel Shved
*Institute for System Programming, RAS*
*email: shved@ispras.ru*

Denis Silakov
*Institute for System Programming, RAS*
*email: silakov@ispras.ru*

## Abstract

*A shared library is a file that contains library code and data in binary form. Application built against the library references the data via symbols and the contents of what's being referenced get known only during the application startup. Library is shipped with header file(s) the program is compiled with.*

*The problem of the binary compatibility (sometimes called „backward compatibility") arises when the new version of library is installed into system and the program, having not been recompiled, is attempted to run in the environment with the new library. The incompatibility may result in fatal errors during the startup or even during the runtime.*

*In this article we deduce the rules that must be followed in order to keep the binary compatibility of a shared library.*

*Unlike most of researches in this area, we also assume that the library may contain its own restrictions upon its usage, more powerful than restrictions of C++ language itself. So the possible restrictions are analyzed as well, and we attempt to weaken the rules of binary compatibility when such restrictions are enforced.*

*As a conclusion we list the rules a programmer should follow to keep his or her library binary compatible. We also conclude that possible restrictions limiting the use of library allow to weaken these rules in relatively small number of cases.*

*For the purpose of this study, we create formal notation for the process of building and using a library, introduce formal definitions of source and binary compatibility and of program behavior. We base the assumptions about mapping source code entities to binary level on the Itanium C++ ABI standard, which describes `gcc`'s way of emitting binary code and data.*

## Index Terms

*Languages, Software libraries*

## 1. Introduction

### 1.1. Premise

C++ nowadays is one of the most popular programming languages [1]. It *is* complex, but the complexity made it become one of the most powerful multiparadigm language, that can combine low-level efficient code with high-level concepts of generic and object-oriented programming.

Like any serious programming language, C++ needs the concept of „library". One of the doctrines of C++ design was compatibility with C language at the source level (with a few exceptions only) and with interfaces of C shared libraries. The decision was necessary to make language more popular by ability to re-use of existing C code [2]. But the questions of C compatibility is still urgent since C is also one of the most popular programming languages ([1]). So, the concepts of C++, that are absent in C, use the same binary interface architecture: the memory the object file occupies is addressed with strings—„symbols". Their semantics is (as in C) not described in binary file, but rather is enforced at compile-time by the header file of the library.

But, compared with C++, libraries have notable peculiarities. They can contain the code of a subroutine provided either at source or at binary level (some libraries, like `boost` or `stdc++`, contain few binary code; most of code is inlined and instantiated in compile time). Further, C++ concepts are much more complex than C's ones, hence the semantics of data being referenced by plain old symbols is also more complex.

Sooner or later, each library faces the question of compatibility with previous versions. For compiled languages the library author should maintain not only source compatibility (i.e. the ability to successfully compile same program code with the new library headers), but also binary compatibility, which lets the program to run with the new binaries of the library without recompilation and to produce the same results as with older version.

Maintaining binary compatibility is a complex task, and it's also important as it takes considerable time to generate effective low-level code out of C++ source. But new versions of popular libraries are released quite often, because every library has numerous bugs that are to be fixed, improvements that are to be implemented and new concepts that allow to create better applications in the future. And the popularity of the library also means that there exist many applications, that use it. They all would have required recompilation if binary compatibility would break with every new release. Furthermore, in the new environment applications may also work differently, but their developers usually promise fixed behavior in advertisements and documentation.

Hence the binary compatibility problem is very important

Table 1. Special symbols used in this paper

| | | |
|---|---|---|
| $\hookrightarrow$ | $A \hookrightarrow B$ | $A$ class is an (indirect) base of $B$ |
| $\mathcal{B}$ | $C.\mathcal{B}$ | set of all bases of $C$ |
| $\mathcal{B}^V$ | $C.\mathcal{B}^V$ | set of all direct virtual bases of $C$ |
| $\mathcal{P}$ | $C.\mathcal{P}$ | most derived primary base of $C$ |
| $\mathcal{V}$ | $C.\mathcal{V}$ | array of $C$'s virtual tables. |
| $-$ | $C.to - C.from$ | offset |

for C++ libraries. However, popular articles on this question are not complete. Instead, they introduce mere list of rules one should follow, the rules usually coming out of practice and the completeness of their set never being claimed. No wonder that these lists differ in such researches. No article also takes into account that, apart from the restrictions of C++ language, that are expressed in header files, a library can have additional restrictions, imposed in free form by documentation.[1] It is possible for these additional restrictions to widen the set of compatible library code.

This article aims to fill these gaps and use a formal approach to binary compatibility problem. We first describe the compiling and building process of library and applications that use it, then we study how the new environment affects the application behavior. It is GNU/Linux system and its default instruments: `gcc` compiler(„GNU Compiler Collection") and `ld`, „The GNU Linker". The toolset being used is chosen to be best specified and popular enough ([6], [7]).

We use [4] as a formal ground for binary analysis; `GCC` follows this standard.

We will not study the questions of inter-compiler and inter-platform compatibility. Also we keep aside compatible exception handling (as most of libraries do not use them). What we will concentrate our forces on compatibility of classes and functions, which code is placed into object files.

## 1.2. Notation

To denote the objects discussed we decided to mix mathematical formalism and the notation used in object-oriented programming. To denote subobjects of compound objects we use C++ notation, where $L.c$ means $c$ subobject in $L$. The same rule holds for member functions; for example, $x.f(y)$ may be treated as $f(x, y)$.

Finally, the table that shows how special symbols are used and what they mean is shown on figure 1.

## 1.3. Basic Concepts

**1.3.1. GNU/Linux System.** In this section the objects involved in program life cycle in GNU/Linux system are described in a formal way. In outline, there are compiler, assembler, linker and underlying semantics of objects they manipulate, regardless of whether they reside in RAM or on disk.

1. Sometimes these restrictions are considerably significant. For example, destructors of widget classes in `Qt` library, are called during the destruction of parent widget. That forces programmer to create widgets with `new` operator only and prevents user classes from having widget members.

**1.3.2. Compilation.** Let's mark out the stages involved in compilation and running of a C++ program.

Source code of an application consists of several *source files* (*cpp* files). Some of them have relevant *header files* describing their interfaces. Each source file may include headers both from the application in subject and from other libraries. Then each *cpp*-file $f.cpp$ is preprocessed and stored in RAM. The preprocessed code, being then independent of other program entities or environment, is compiled, resulting in the *object file* $f.o$.

We combine these stages and will write as follows:

$$f.o = f.c.compile(f.h,\ lib.h).$$

As arguments this *compile function* takes header files $f.c$ includes. Indeed, if $f.cpp$ contains **`#include`** `<lib.h>` line, then preprocessed source will change subject to $lib.h$ contents. This change is represented in a natural way—as an argument to function. For example, $f.c.compile(f.h,\ lib'.h)$ denotes that *the contents* of header file included via the particular line with „include" statement are changed, its name acting as reference only and being invariant.

After all sources are compiled, the work flow proceeds to *linking* stage. Its result is an *executable file* $f.exe$, in which the code of all object files and statically linked libraries is comprised. We denote it as

$$f.exe = f.o.link\,(f_1.o,\ f_2.o \ldots f_n.o)\,,$$

where $f.o$ is an arbitrarily chosen object file from the set of all ones being compiled.

However $f.exe$ still doesn't ultimately define the code that will be executed. If the application was compiled with a *shared library*, then, at the start of its execution, dynamic loader's code is run. Its purpose is to link the symbols used in application executable as references to the actual code and data addresses in RAM. The libraries to be loaded dynamically are chosen from the current environment via sonames and may differ from the ones linked to during preparing the application executable (see [5] for details). Therefore, similarly to the compile function notation, the dynamic linking function should also take shared objects as arguments: $f.exe.dynlink(lib_1.so,\ lib_2.so,\ \ldots)$.

The result of running an application and its consequent possible dynamic linking is *application context*. The context then consumes *input data* and produces *output data*. „Input data" comprises very broad range of data: OS state, sequence of random numbers returned by corresponding functions, command line arguments, filesystem state and environment variables. In other words, input data unambiguously define output data.

The application context obtained is therefor a map from set of all input data to set of output data. We denote it like an usual function:

$$f.exe(input) = output \qquad (1)$$

$$f.exe.dynlink(lib.so)(input) = output \qquad (2)$$

**1.3.3. Notes on Specifications.** Talking over binary compatibility, not only we require the application to run in new environment, but expect it to yield the correct results. The most common way to define correctness of the behavior is to use concept of *specifications*. Program behavior is described by its context, which is a map from input to output. So, it's natural to define *specification of a program* as map from *correct input* to output, the mapping being *conformant* to library code.

*Correct input* is an input that doesn't cause precondition violation [2] for shared library functions, as if abstract C++ machine defined in C++ standard [3] is run.

By *conformant mapping* we mean the mapping, that executing the library functions on abstract machine in such way, that their return values and side effects are conformant to the relevant library specifications.

Therefore we can define, that *program conforms to specifications* („works correct") if its context and specification, restricted to the set of correct inputs, are equal. We denote it like this:

$$f.exe.dynlink(lib.so) \overset{\cap}{=} spec \tag{3}$$

Contract library specifications may govern not only function calls but virtually every way of using of concepts defined in header files. The consequences were discussed in .

A program or a library may be „badly written", i.e. have empty set of correct inputs. In further analysis we only take „well written" libraries and programs into account, their set is denoted as $\mathbb{P}$.

**1.3.4. Shared Libraries.** Let's define our subject. A *shared library Lib* is a structure

$$Lib = \langle H = \{h_i\}, CPP = \{cpp_j\}, so, soname, spec\rangle,$$

where

- $h_i$ are header files;
- $cpp_j$ are source code files that contains definitions of (some) functions and variables declares in header files;
- $so = link(..., cpp_j.compile(h_{j_1}, h_{j_2} ...), ...).makeso$ — an object file compiled in a special way, which can be used as an argument for dynamic linking function;
- $soname$ is a library name via which the *so*-file is found by dynamic linker;
- $spec$ is an encapsulated object with the sense discussed in 1.3.3.

The definition requires *cpp* files to comply with header files declarations—i.e. compile without errors.

## 2. Binary Compatibility

### 2.1. Definition

To simplify the further study we will consider program $p$, that consists only from one source file, and a library $L$, that also consists from one source and one header file.

2. a library documentation practically doesn't use the word „precondition". But its sense is comprised in phrases like „the value of this parameter *shall be* more than zero" or „the behavior for NULL pointer *is unspecified*".

Table 2. Symbol versioning example

| $A'.h$ | $B'.h$ |
|---|---|
| `namespace libA{`<br>`#include <A.h>`<br>`}`<br>`using namespace libA;` | `namespace libA{`<br>`#include <A.h>`<br>`}`<br>`namespace libB{`<br>`#include <B.h>`<br>`}`<br>`using namespace libB;` |

Let's assume that a new version is released — $L'$. Then $L'$ *library is binary compatible with* $L$, [3] if

$$\begin{aligned}\forall p \in \mathbb{P} \rightarrow & p.cpp.compile(L.h).link(L.so). \\ & dynlink(L.so) \overset{\cap}{=} \\ & \overset{\cap}{=} p.cpp.compile(L.h). \\ & link(L.so).dynlink(L'.so)\end{aligned} \tag{4}$$

If we would recompile program in the new environment its context would look like that:

$$p.cpp.compile(L'.h).link(L'.so).dynlink(L'.so).$$

The behavior in new environment may hence differ; that's what we are to evade.

The definition of binary compatibility doesn't really constrain itself to the cases where it's useful. For example, there's no obstacle that prevents us from investigating the issues of compatibility of math library and window manager library. Therefore we should lay down one more condition, namely *source code compatibility*.

*Library* $L'$ *is source compatible with* $L$, if

$$\begin{aligned}\forall p \in \mathbb{P} \rightarrow & (\exists p.o = p.cpp.compile(L.h)) \\ & \Rightarrow \left(\exists p'.o = p.cpp.compile(L'.h)\right).\end{aligned} \tag{5}$$

Later on we will discuss only binary compatibility of source compatible libraries.

### 2.2. Symbol Versioning Approach

Let's examine first the following method referred to as *symbol versioning*. We will now show that any libraries may be made binary compatible even if they're of different kind. Indeed, assume $A$ and $B$ are libraries. Then let's create $A'$ library, that's source compatible with $A$, and such $B'$ source compatible to $B$ that it is binary compatible with $A'$.

To achieve this we keep *cpp* files intact, and link the objects files into single *so* file, having rewritten the header files as shown on figure 2.[4]

The libraries produced are binary compatible and same programs may be compiled with them. Source compatibility between $B'$ and $A'$ is equivalent to that between $B$ and $A$.

However this approach suffers from the code duplication.

3. we say that the *new* version is compatible with the *old* one, and not vice versa

4. `C` language doesn't have namespaces, but `gcc` introduces the other symbol versioning mechanism for C language; see [5]

Note that symbol versioning can not be used with every bugfix release, as each bugfix forces us to create a new function instance. Symbol versioning is used (for example, in `glibc`) with major releases, the functions in bugfix releases sharing the same version and having to be binary compatible.

To prevent duplication we might use aliases for duplicating code and data. But how can we be sure that code is duplicated? The question of whether the code of function is duplicated with version change is essentially equivalent to the question of binary compatibility. Therefore, symbol versioning doesn't govern all binary compatibility problems and further study is necessary.

## 2.3. Causes of Binary Incompatibility

There are different ways to build an run application:

$$p.cpp.compile(L.h).link(L.so).dynlink(L.so) \quad (6)$$
$$p.cpp.compile(L.h).link(L.so).dynlink(L'.so) \quad (7)$$
$$p.cpp.compile(L'.h).link(L'.so).dynlink(L'.so) \quad (8)$$

Contexts (6) and (7) differ only in dynamic linking function arguments, i.e. in the set and internals of the symbols the library exports. In `C` language it only means that during the calls to the functions with external linkage the other code will be executed. However, C++ libraries contain more symbols than those of functions and static data, and some of auxiliary symbols are used in an unobvious way. Binary compatibility is caused by the difference between symbol sets and contents in different versions of library.

According to the definitions of building the application the object file $p.cpp.compile(L.h)$ already contains information about symbols it might need in the shared library. However, an only source of information about the library so far is header file $L.h$. Therefore, *the set and internal structure of symbols exported are completely defined by header file*. The source file $L.cpp$ only defined internal code and *some* data these symbols point to.

However, during the linking with $L.so$ no code or data defined in $L.cpp$ is inlined into $p.exe$. This code is loaded on dynamic linking stage only. But at this stage the application requires the presence of symbols defined in $L.h$, but the library $L'.so$ the program's being linked to provides symbols, defined by $L'.h$. That's one of the causes of binary incompatibility.

Furthermore, if a code defined in *cpp* file is called via symbol, the callee will consider that arguments are laid out like in callee's native header file, $L'.h$. However, the caller assumes that all library data is laid out like in $L.h$—the file the program was compiled with. That's an important matter, because the semantics may stay unchanged from the user point of view (summarized in C++ „source" standard [3]), it could change from binary point of view (ABI standard [4]).

We can now divide the causes of incompatibility into the following groups:

1) **implementation change**, the case when constant or function whose signature remains intact, changed its definition;
2) **compiled code notions incompatibility**, the case when the notion the function in *cpp* file has of layout of its arguments has changes due to alteration of declarations in header file. These alterations only depend on $L.h \rightarrow L'.h$ change and may be studied separately from 1.
3) **altering or removing of special symbol**, which definition is deduced by the compiler based on header file. The data the symbol refers to could be used in application binary code as in notions given by $L.h$, but handled in $L'.so$ as in $L'.h$.
4) **errors during dynamic linking**, caused by absence of definition of an entity from $L.h$ in $L'.cpp$ hence $L'.so$.

## 3. Study of Incompatibility Sources

Let's thoroughly study the sources described in 2.3.

### 3.1. Errors During Dynamic Linking

After the application is built against $L.so$, no static linking errors can arise. However, at the time it's run dynamic linker tries to link it with $L'.so$ instead (because $L.soname = L'.soname$), so it may lack symbols the application references and abnormally terminate before it has any chance to run its code.

Dynamic linker searches all *external dependencies* of $f.exe$ in the libraries loaded. They're fixed at compile-time and marked at link time as external, when the linker finds definition of symbol in a shared library rather than in static one or application's object file. Here the linker ensures that all dependencies are satisfied. The dependency may be global variable's symbol, non-inline function, non-template function of fully-instantiated template function.

We well use term „use symbol" instead of „use declaration the symbol relates to".

We should note that, among external dependencies, there can be the symbols allowed to be used in userspace code. *Userspace code* is a code that compiler takes as an input. It therefore contains $f.cpp$ and $L.h$; and inline functions defined in $L.h$ in particular. The rules for keeping compatibility will be formulated in terms of userspace code, so ***the library developer must follow the rules in his own inline functions code as well***.

We should also note that for a member function call not only its own symbol may be required, but also *virtual table* (vtable) may be involved for virtual functions and for non-virtual member functions of virtual bases calls. Among the other info (see 3.4.2 for details) it contains pointers to virtual functions definitions as symbols, that are resolved in runtime.

$L'.cpp$ may lack symbol described in $L.h$ only because that symbol is not deduced from $L'.h$ (by the property of shared library definition — see 1.3.4). Among explicit non-auxiliary symbols only non-inline functions and global data present in header file. If one of such symbols is withdrawn, and it can be used in $p.exe$, $L'$ would then be incompatible with $L'$. As it is obligatory to explicitly use such symbol

Table 3. Similar functions but different external names

| $L.h$ | $L'.h$ |
|---|---|
| **typedef int** Type;<br>**void** function(Type); | **typedef float** Type;<br>**void** function(Type); |

for dependency to appear, the incompatibility of library is equivalent to the plausibility of this symbol usage in userspace.

Therefore, ***it's impossible to withdraw from header file a symbol that is allowed to use in userspace code***.

A symbol name is constructed out of its declaration through use of „mangling". The mangling procedure is fully described in section 5.1 of [4]; here we only outline some key conclusions.

A symbol (external) name for a declaration for `GCC` compiler is uniquely and unambiguously defined by the conjunction of the following properties:

1) **filly-qualified name of declaration**, the name of declaration and fully-qualified name of enclosing scope (class or namespace)[5];
2) **vector of argument types and vector of template parameter types (for instantiations)**. The types with all typedefs substituted are considered for this purpose, however, structures, classes and unions are not expanded—their name is taken only. For example the functions shown on the figure 3 have different external names.
   All fully-substituted types are encoded in unique way, the types and member functions being also distinguished by cv-qualifiers.
3) **return type of a template function**. If template function is instantiated, its return type is also encoded into external name.
4) **set of function's thunks**. For functions that require adjustment to **this** pointer (overloaded functions of non-primary base, separately for virtual bases, virtual base subobjects[6] and all other bases) or to return value (for covariant return types) special entry points are created and then used in the function call algorithm. Special functions called „thunks" for different ways to call the functions are emitted into object file[7].
   The application, that doesn't derive the defining class, can't have them as direct dependencies, because the calls to them can only be encountered through use of vtable. But if application is allowed to derive a class with such functions and it actually does, then the use of them in derived class' members definitions will require adjustment and henceforth the relevant dependency.
   The set of thunks define the rules of confronting the function call operator and the symbol that references the actual code. Therefore, if the number of entry points or the causes of their appearance is changed, it should be encountered as symbol name change. [8]
   Let's formulate the rule: ***let the virtual function `C::f`, be such that $C \in D.\mathcal{B}$. Then the alteration of any of the following properties will cause binary incompatibility:***
   - whether it has covariant return type or, if it has, the return type itself
   - whether $C = D.\mathcal{P}$;
   - if $C \neq D.\mathcal{P}$, whether $C \in D.\mathcal{B}^V$;
   - class $W$, such that for $W$ holds

$$W \in D.\mathcal{B}^V, C \hookrightarrow W \hookrightarrow D \qquad (9)$$

$$W \notin D.\mathcal{P} \qquad (10)$$

$$\forall W' \in D.\mathcal{B} \rightarrow \Big(C \hookrightarrow W' \hookrightarrow W \Rightarrow W' \notin D.\mathcal{B}^V\Big) \qquad (11)$$

   Compatibility issues that arise from thunks' contents are discussed in 3.4.1.

So, if a property of the definition alternates, the library loses binary compatibility. One of the most stunning examples of it, described in [8], is when one adds default argument, the mechanism initially designed to keep compatibility (unluckily, source one). Indeed, when you add a new default argument to the function, its vector of arguments changes and hence the external name changes as well.

In this list the property „whether function is inline" is absent, because there's no external names for inline functions at all. However, many developers do treat it as function property, so here's the rule for that: *to keep binary compatibility the **`inline`** qualifier must not be added to the function allowed to be called from the userspace code.*

## 3.2. Implementation Change

New versions of libraries are released to add the new or to remove the obsolete functionality, fix bugs or improve the underlying implementation algorithms. From a newbie's point of view bugfixes do not cause any harm. because it hardly changes anything, especially in the set of external names. From a formal point of view only the improvement of existing algorithm is insignificant as long as it doesn't change the contract of function. What we call „a bugfix" is actually a change of specification and a confession that $L$'s specification is not $L.spec$ defined by help files, but something else, and only $L'.spec$ made true specifications coincide with the alleged one.

When developer changes symbol specifications, the application calls the new implementation, namely the one in $L'.so$ and may change behavior, what leads to losing binary compatibility. It also may not.

Consider the following example. $L$ contains a `streq` function, that compares strings. $L'$ introduces a new feature to compare them case-insensitively, if the global trigger variable **`bool`** `case_insen` was set to true, its initial value being false. The specifications and behavior of `streq()`

5. treat this as recursive definition
6. the term „morally virtual" is used to name such classes
7. they don't have to be separate functions, sometimes they're merely different entry points into single piece of code
8. The set of thunks may be treated as a single „multisymbol" for the current one.

are changed, but no program compiled with $L$ is capable to use the new $L'$'s functionality and actually yield improper behavior.

***The change of function implementation doesn't lead to binary incompatibility iff the new functionality is unreachable from any correct executable linked to prior library version***.

From this point of view bugfixing *is* a binary noncompliance. The program will work better, but in the other way. Sometimes bugfixes in $L$ cause errors in $p$, if $p$ implemented a workaround for the bug fixed and it became broken with the new version[9]. However bugfixes are considered useful rather than harmful, because the abstraction of the code into third-party libraries intends to separate the workflow of the application and of the code it uses as backend.

We should note that sometimes binary compatibility is understood as the ability of a program to behave in the new way in the environment with the new library version. Of course, from this point of view, bugfixes do not affect binary compatibility.

However we think that this approach is incorrect. First of all, the part of $L$ and the notion about it is anyway inlined into the application's executable (as a side effect it could even reduce the number of compatible libraries in the other cases, if the alternative definition of compatibility is used). Secondly, as it was said in the intro, an application developer should fix the behavior of the program in a help file of sorts, so the library changes would reduce the separation of application and underlying library.

## 3.3. Compiled Code Notions Incompatibility

In this section we will look for the such ways of altering the header file, that cause $L'.cpp$'s code, that implements data access accordingly to $L'.h$'s notion, to access the same data that are laid out as in $L.h$. In the other words, we will study the raw memory layout semantics and how to prevent incorrect access to it.

We can separate two different directions:

1) **access to the library's memory from the userspace** through global variables and specifications-compliant operations with them;
2) **the access to the application-allocated memory from the library**; the memory's having been allocated for the library's types declared in $L.h$ through functions in $L'.so$.

In the point 1 we mean direct access to global variables. Indirect access to the memory through interface functions is under the library's control and doesn't cause incompatibility directly (i.e. is studied somewhere else). Just as well, calls to class static members or global variable's members don't cause incompatibility immediately.

All operations considered are equivalent to reading the variable of integral type and to direct writing to it or to complex structure as a whole. When recording the structure, if its copy implementation is deduced by compiler (i.e. the copy constructor is not overloaded) only semantic violation errors my arise. Therefore, ***binary compatible access to variable, that may be accessed from the userspace for reading and writing, is possible only if the alignment of $T'$ coincides with alignment of $T$ on the first $sizeof(T)$ bytes.*** In the other words, you only can add fields to $T$ class, but you can't alter the ones defined in $L.h$.

The analysis of 2 should be more elaborative.

It differs from the point 1 by the capability of library to call its own types with arbitrary class of operations that's broader (at least not more narrow) than what is possible from outside the library. The ultimate principle can be formulated like this: *$L'.cpp$ should be created in the way for it to be able to distinguish the origin of the data being handled, whether it's $L.h$ or $L'.h$.* [10] This approach is equivalent to symbol versioning, which, as shown in section 1.3.4, still requires further study of what can be done without it.

Let's assume that a function takes one value as an argument (member functions are implemented as simple `C`-like functions that take pointer to **`this`** as its first argument), the type (possibly, indirectly, via pointers and references) depending on $T$ type, declared in $L.h$ and $L'.h$. Let's call $T'$ what $T$ became in $L'$.

If the argument is passed by value, then in binary compatible application $T = T'$, because these types should have equivalent sizes (as the memory in stack for them is allocated via caller) and semantics on first $sizeof(T')$ bytes. Therefore, *only the types that are passed via pointer/reference to the library functions may be altered.*

The rules from 1 are applicable also to the types, that are returned by value from library routines, because caller is responsible for copying values back from stack.

If object of type $T$ is passed by reference into the function, that expects reference to $T'$, it can control the access to memory access within the object.

However, if it's possible for $T$ to be allocated into automatic storage in userspace (that includes being base class or class member), the pointer to it may address less memory than $T'$ denotes; the behavior being undefined upon access to it. Moreover, if a class has an explicit constructor, its call may lead to memory access violation. Therefore, the following rule holds: ***the increase of size of a class that can be allocates in the automatic storage in the userspace causes binary incompatibility if the functionality, that accesses the new memory, is reachable keeping the conformance to „old" library specifications***. Note that new members may not extend class size; that's especially notable for bitfields (see [8] for bitfields as a technique to maintain compatibility).

Note also, that class size may be increased not only with new members, but with new bases as well. If such new class requires more memory (it may not for a nonempty, but relatively small class; however it may require for an empty class as well; refer to [4]), then new memory is laid out

9. The known example nowadays is a bug in `Qt` 4.4 fixed in version 4.5, but many `KDE` 4 applications contained a workaround and become incorrect

10. as an example, one may require to explicitly specify the version of library used in the application by the special function call or global variable or something else.

before any of the members, all data members shifting. For a class with at least one accessible data field that causes incompatibility.

Therefore, practically, the paragraph above means that *adding a new base that increases the size of the class leads to binary incompatibility*.

There are several techniques that allow adding functionality to the class maintaining binary compatibility: „d-pointer", described in [8], techniques that emulate interpreted languages elements (see [8], „Adding new data members to classes without d-pointer"). One can avoid problems by disallowing to allocate memory in automatic storage in userspace, hence forbidding to derive the class, but that undermines the basis of OOP and has limited use.

The conclusion follows: ***The change of class hierarchy (except cases when it involves the change of size of no bases), change of members' order, size and increase (change, in case the class can be passed to or returned from library function by value) of their amount leads to binary incompatibility***.

To apply this rule into practice you might want to experiment with size of your structures, but that's hardly will be useful. Empty bases (and these are nearly all bases that don't make class change) are best served virtual and the restrictions on virtual bases are more strong, what you will see in section 3.3.

## 3.4. Auxiliary Symbols Change

Along with symbols described in 3.1, GNU C++ compiler adds auxiliary symbols to binary level. They are used in virtual functions call algorithm (vtables), expose support for low-level inheritance-related code generation (VTT, several variants of constructors and destructors) and several thunks.

This section investigates the class of $L.h$ alterations that don't cause binary incompatibility through auxiliary symbols alterations. The key problem is that part of C++ low-level code support mechanisms are generated in compile-time and the symbols are assumed to be laid out as in $L.h$. During the dynamic linking the application references these symbols in the way described in $L'.h$, what causes incorrect program behavior. The bodies of these symbols also can change, but part of these changes, namely $L.cpp \to L'.cpp$, has already been studied in 3.1 and 3.2; we will elaborate the other part here.

Let's study how each symbol type influences the compatibility.

### 3.4.1. Symbols Introduced via Functions.

- **Constructors and destructors** cause compiler to emit symbols for compete object constructor with and without memory allocation and for base object constructor (for construction of non-static members and non-virtual bases); the same symbols are created for destructor (but with freeing the memory instead of its allocation). These symbols should have played an important role in binary compatibility, but unfortunately it's not always possible to encapsulate memory allocation procedure in the shared library. Therefore the possibility of compatibility violation is restricted to matters discussed in 3.3, and to that, upon withdrawing of all virtual bases (which causes incompatibility, as we will see in 3.4.2), relevant symbols also disappear from library, causing link-time error.
- **thunks**. Thunks are described in section 3.1. Here we will assume that set of thunks didn't change; only actual offsets could.
  Part of these values, that concern **`this`** adjustment, are described in 3.2.3 section of [4]. It clearly shows that adjustment is done with different offset values, that, as described in section 5.1.4 of that standard, are encoded into external names of thunks, examples shown on figure 4.
  According to comments in `gcc` code[11], entry point body only depends on these values, on whether covariant type presents and on external name of the function the thunk is associated to.
  In the other words for thunks with same names equivalent bodies are emitted (limited to the possible discrepancy in the actual function's body).
  As any shift of classes through the hierarchy is incompatible (see 3.3), *thunk bodies will coincide in $L.so$ and $L'.so$ if the other binary compatibility conditions are held.*

Table 4. Thunk names

| | |
|---|---|
| `_ZThn8_N7Derived3virEv` | non-virtual |
| `_ZTv0_n48_N7Derived3virEv` | virtual |
| `_ZTvn8_n48_N7Derived3virEv` | virtual+vbase offset |

**3.4.2. Vtables.** In this section we will use concept of *class hierarchy*, the tree, that depicts the class' direct bases, then their bases and so on, the edged representing direct derivation. Some rules will also be formulated in terms of class hierarchy. The developer should remember that ***when the hierarchy of $C$ is alternated, hierarchies of some classes that derive $C$ may also change*** (and most of them will, unless the change is adding a new virtual base that already presents in all classes derived from $C$ before it in preorder). Practically that means that without additional internal requirements to inheritance (we've just outlined one possible rule in parentheses) these rules are useless and it's best to prototype and check them, or apply to classes which derivations developer can control.

Vtables is a structure that supports virtual function call algorithm, virtual base access and RTTI for **`dynamic_cast`**. In C++ all static types are known in compile time, therefore virtual tables are referenced through pointers, each for every primary base group. Every pointer references some data placed in the translation unit, where first virtual function body is emitted.

Vtable structure is fully described in [4], section 2.5. We will only give some general information.

11. check the description in files `gcc/cp/cp-tree.h`, line 3317, and `gcc/cp/method.c`, `make_thunk();` function.

Table 5. Vtable group layout example

| Classes | | | | Entry | Offset |
|---|---|---|---|---|---|
| $C$ | | | | vcall offset for $B_2 :: f_3$ | $-72$ |
| $C$ | | | | vbase offset for $D_1$ | $-64$ |
| $C$ | | | | vbase offset for $B_1$ | $-56$ |
| $C$ | $B_3$ | | | vcall offset for $B_2 :: f_3$ | $-40$ |
| $C$ | $B_3$ | | | vbase offset for $B_1$ | $-32$ |
| $C$ | $B_3$ | $B_2$ | | vbase offset for $B_1$ | $-24$ |
| $C$ | $B_3$ | $B_2$ | $B_1$ | offset-to-top (zero) | $-16$ |
| $C$ | $B_3$ | $B_2$ | $B_1$ | RTTI (of $C$) | $-8$ |
| $C$ | $B_3$ | $B_2$ | $B_1$ | $B_1 :: f_1$ | 0 |
| $C$ | $B_3$ | $B_2$ | $B_1$ | $B_1 :: f_2$ | 8 |
| $C$ | $B_3$ | $B_2$ | $B_1$ | $B_2 :: f_3$ | 16 |
| $C$ | $B_3$ | $B_2$ | | $B_2 :: g_1$ | 24 |
| $C$ | $B_3$ | $B_2$ | | $B_2 :: g_2$ | 32 |
| $C$ | $B_3$ | $B_2$ | | $B_2 :: f_3$ | 40 |
| $C$ | $B_3$ | | | $B_3 :: h_1$ | 48 |
| $C$ | $B_3$ | | | $B_3 :: h_2$ | 56 |
| $C$ | $B_3$ | | | $B_3 :: h_3$ | 64 |
| $C$ | | $D_2$ | | vbase offset for $D_1$ | $-24$ |
| $C$ | | $D_2$ | $D_1$ | offset-to-top (nonzero) | $-16$ |
| $C$ | | $D_2$ | $D_1$ | RTTI (of $C$) | $-8$ |
| $C$ | | $D_2$ | $D_1$ | $D_1 :: g_1$ | 0 |
| $C$ | | $D_2$ | $D_1$ | $D_1 :: g_2$ | 8 |
| $C$ | | $D_2$ | | $D_2 :: h_1$ | 16 |
| $C$ | | $D_2$ | | $D_2 :: h_2$ | 24 |

Let's consider vtable group $C.\mathcal{V}$ of $C$ class. They're laid out consequently, in the same order the base classes are placed in $C$'s body. Every vtable $V \in C.\mathcal{V}$ is aligned around the point of origin referenced by $C.ptrto(V)$; it relates to the primary base group $B_1 \hookrightarrow B_2 \hookrightarrow \dots B_n$. Immediately before the zero, with negative offset, RTTI pointer and $C - C.ptrto(V)$ offset (*offset-to-top*) are places. Then, for $i := 1..n$, are appended the pointers to final virtual function overriders of the functions first introduces into $B_i$. To the beginning, at negative offset, if $B_i.\mathcal{B}^V \neq \emptyset$, offsets $B_i.\mathcal{B}^V_j - C.ptrto(V)$ (vbase offsets) are appended; the less $i$ is, the closer to point of origin offset's placed. Then, for each virtual function, declared in base $S$ of $B_i.\mathcal{B}_j$, such that $\forall D : K \hookrightarrow D \hookrightarrow B_i.\mathcal{B}_j \Rightarrow D \notin \mathcal{V}$, and that it's finally overridden in $K_k$, the offsets $B_i.\mathcal{B}^V_j - K_k$ (vcall offsets) are appended in the same way as vbase offsets, but after them in the „negative" direction.

So the information about primary bases is „sliced" so they share vtable and vtable for base class is, at the memory such vtable is allowed to access, coincides with vtable as if it was allocated separately. See the example of vtable layout on figure 5 (the hierarchy is $B_1$ is a primary virtual base of $B_2$, which is primary base of $B_3$, which is primary virtual base of $C$, which also derives $D_2$, which has a primary virtual base $D_1$; virtual function $f_3$ is defined in $B_1$ and overloaded in $B_2$).

We can see, that pointers to functions, offsets and data pointers present in the vtable. It's irrelevant if they all have different sizes because they can't intermix due to carefully elaborated vtable layout. However as vtable group is referenced by only one symbols, the size of each can't change, as offsets from the first vtable to any other$C.\mathcal{V}_i$ are precompiled in $f.exe$.

Hence $C.\mathcal{V}$ depends on mutual location of virtual base groups, on the order of classes within these groups and on their virtual functions.

Therefore an only $C.\mathcal{V}$ change possible is to add new vtables or extend the last one in the group ($C.\mathcal{V}_{|C.\mathcal{V}|}$). However, if that's the table of $V$ such that $V \in C.\mathcal{B}^V$, developer can't add virtual functions overridden in derived classes as it would ass vcall offset shifting table the shift to which is precompiled. But unfortunately the function will most likely be added to virtual base as their vtables are places at the end of $C.\mathcal{V}$.

Furthermore, when you derive the class in the userspace, the derived vtable is constructed like in $L.h$, but the library functions will implement virtual functions call algorithm according to $L'.h$ notion. So, ***extension of vtable is impossible without compatibility loss, if extending the table functions or classes are used in userspace code.*** We will assume that this rule holds. Let's study the ways of table extending.

1) **New base class**
The extension may be achieved by increasing the amount of dynamic bases; whenever a class made virtual or a first virtual function is added to one of them. But the new class can't be virtual as it would add vbase offset to the beginning of vtable. As change of class mutual interposition is forbidden as well, only two options remain.
    a) Add class $N$ that will share virtual table $C.\mathcal{V}_{|C.\mathcal{V}|}$ with other classes. This would just add a new „slice" to the vtable. Such a change is only compatible when nothing would be added to the „negative" side of vtable. That's in turn possible only when non-virtual dynamic class is added, the class having no virtual derivant (otherwise virtual functions of $N$ would cause new vbase offsets). Non-virtual class, that doesn't define virtual functions is not dynamic, therefore ***an only compatible way to add a class sharing the last vtable in the group, is adding a nonvirtual base in case when*** $C.\mathcal{B}^V = \emptyset$
    b) Add class $N$, that yields new vtable in the group. As it's non-virtual class, $C.\mathcal{B}^V = \emptyset$; and $N$ will be added after all dynamic classes in preorder. The overload of virtual functions by other classes will be studied in point 4, and the conclusion will be that it won't cause incompatibility. Therefore, ***non-virtual dynamic class can be added to the end (in preorder) of hierarchy of class without virtual bases***.
2) **Adding a completely new virtual function**
Let a virtual function be added to subclass $B$, the virtual function being added not overriding and being overridden by any other function. It can only be added to the very end of vtable group, i.e. to the most derived class of this group. Such an addition can't cause vcall offset only if $B$ is virtual and $B$ doesn't have virtual bases. However, if $B$ does have virtual derivants, $C.\mathcal{B}^V \neq \emptyset$, i.e. $B$, as most derived class of

$C.\mathcal{V}_{|C.\mathcal{V}|}$ group, is virtual base itself, what proves that new vcall offsets would never be added.
Therefore, ***adding not overloaded and not overloading virtual function to the end of the most derived class of the last group of virtual base doesn't break binary compatibility***.

3) **Withdrawing a virtual function**
The withdrawing of function may retain compatibility if adding of it to the resultant classes causes appending to the very end of virtual table. Of course, the same rules as in 3.1 apply to the function being withdrawn, if the function can be called from the userspace. But in some cases virtual function's symbol isn't referenced directly, so there's no external dependency on it.
Okay, let's assume that the function is called through virtual table. If the class can be derived in userspace, then the new vtable is created for it at compile time and, upon the call of the virtual function in subject through that vtable, it will fail. But then the call will fail in runtime due to absence of the proper symbol.
Therefore an only conclusion possible is that ***withdrawing a virtual function (and making nonvirtual a virtual function[12]) breaks binary compatibility***.
4) **Adding an overloading virtual function**
Let the new function $K :: f$ be such that it overloads a (probably pure) virtual function of some base class. In case of covariant overloading it requires a new vtable entry; point 2 applies in this case. Otherwise it's required to overwrite all pointers to function by replacing them with the relevant entry points; this doesn't cause binary incompatibility. An only condition remaining is for a function not to add new vcall offsets. Therefore, ***adding a virtual function to $K$ class doesn't cause binary incompatibility iff the function doesn't covariantly overload and $\forall V \in C.\mathcal{B}^V \Rightarrow K \notin V.\mathcal{B}$.***
5) **Adding a function becoming overloaded**
Assume a function is added that doesn't overload any other. As this function is added relatively close to vtable's point of origin it can only keep compatibility if its derivants the class is a primary base for do not add more functions to the vtable and if this function doesn't add more vcall offsets (see 4).

**3.4.3. VTT.** VTT[13] is a structure that keeps pointers to virtual tables during construction. These tables do not possess own symbols; it is only VTT they can be referenced through. Each VTT entry refers to a vtable keeping information about the state of object during construction within larger object. The virtual function pointers will only point to the routines of class constructed so far (in the process of complex hierarchy initialization), its RTTI will be proper and vbase offsets will point to virtual bases allocated as in bigger object. Only the latter is the reason of introducing the new entities compared to the usual vtables; therefor it's only classes with more than one (indirect) virtual base who have VTT assigned.

Where VTT is used is construction code for derived classes. Let's consider a class that can be derived in the userspace code. Taking the conclusions of section 3.4.2 into account, we reduce the analysis to adding non-virtual dynamic classes to the hierarchy of class that doesn't contain any virtual bases. However, no VTT is created for such classes. Therefore *study of VTT doesn't yield any results*.

12. C++ rules state that if function $f$ is declared as virtual in $B$ class, then $\forall C : B \in C.\mathcal{B}$, $C :: f$ is also virtual

13. most likely, it's an abbreviation of „virtual tables table"

## 4. Conclusion

### 4.1. How to Keep Compatibility

Let's sum our study up. We have studied enough to formulate the rules the developer is to follow to retain binary compatibility with the old version of shared C++ library in GNU system, assuming that additional constraints in addition to C++ rules apply. We called the code compiled into application *userspace code*; it includes inline functions of library headers and application code itself. The classes that are allowed to be instantiated in userspace code are called *userspace classes*, other classes are *internal*. The functions that can be called from the userspace code are referred to as *userspace functions*.

Let us have $L$ library that we're going to alternate and get $L'$ library, the new version. Then, $L'$ will be binary compatible with $L$, if all following rules apply:

1) **any userspace function with „external linkage" shall retain its external name** (1.3.4). Therefore, you should keep true arguments type as they appear after all `typedef` substitutions and their number. To learn what changes external name, refer to 3.1.
2) **no userspace function may be removed or made inline, either member or global, virtual or non-virtual; no virtual function may be removed even for internal class**. Refer to sections 3.1 and, for virtual function discussions, to point 3 of section 3.4.2;
3) **no function implementation defined in *cpp* file may be changed in incompatible way**, i.e. if user calls new functions in an old way, that must be plausible and behavior must be the same (section 3.2). A special exception holds for bugfixes, but note, that they may break workarounds;
4) **layout of the first $sizeof(T)$ bytes of types of directly accessible userspace global data must be the same**; this holds for both static class variables and for internal classes (3.3, point 1);
5) **the size of userspace class must be the same** if it has non-inline constructors; if all constructors are inline, you should use symbol versioning of sorts to prevent access to new part of the type layout from the new function;
6) **classes in hierarchy of *all* userspace classes must be the same and in the same order** unless the classes being moved through hierarchy are empty bases of non-dynamic class (but you still need an experiment to ensure that sizes are the same). See 3.3 and 3.4.2;

7) **dynamicity of classes in hierarchy of userspace class must be the same except for userspace class without virtual bases, where you can make non-dynamic class after all dynamic classes in preorder** see (3.3 and point 1 of section 3.4.2);
8) **you can introduce new virtual functions overloading the old ones, except for the case of covariant overloading and overloading of function of a virtual base** (point 4 of section 3.4.2). You should be assured that the call to this function will yield the same results as if it were called in a way allowed by $L$ specifications;
9) **a completely new virtual function may be added to the end of the most derived class if its hierarchy doesn't contain any virtual base** (point 2 of section 3.4.2).

## 4.2. Conclusion

We may compare the rules deduced by us and summarized in 3.4.3 with the compatibility guide [8] suggested by the KDE developers and known to be most complete.

We see that our formal approach didn't yield more results than the developers deduced from the practical experience. It appears that requirements for binary compatibility can't be relaxed due to additional restrictions on the library use except for a limited number of cases. Namely, there's more freedom for internal classes, that can't be instantiated in automatic memory or derived by user. However, such classes can't be considered as exhaustively using C++'s OOP flavors, although implement quite a popular „singleton" concept (see [9]).

Therefore we conclude that the current C++ ABI is incapable to provide more compatibility even with additional restrictions upon the use of C++ constructs provided by library's headers.

We should also note that the current `gcc` ABI is influenced by the desire to keep away from inserting elements of interpreted languages into ABI and by „incremental" way of binary representation (the architecture when the most common cases—single inheritance and simple virtual functions—induce more simple and fast binary representation). As a result, complex concepts are both considered unsafe and their uncareful use causes incompatibility.

Perhaps, the other ABI model would better fit compatibility aims, but this question needs further research to discover the best balance between maintainibility and performance, and it also needs careful cost estimation, as the benifits of its change should be greater than the expenses required to adopt it.

## References

[1] *TIOBE Programming Community Index for January 2009.* http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html;

[2] Bjarne Stroustrup. *A History of C++: 1979-1991.* History of Programming Languages conference, 1993;

[3] ISO/IEC 14882:2003. *Programming languages — C++*

[4] "informal industry coalition consisting of (in alphabetical order) CodeSourcery, Compaq, EDG, HP, IBM, Intel, Red Hat, and SGI". *Itanium C++ ABI (Revision: 1.86)*

[5] Urlich Drepper. *How To Write Shared Libraries.* 2006.

[6] B. Guptill, B. McNee. *Booming Support for Mission-Critical Application Workloads on Linux.* http://research.saugatech.com/fr/researchalerts/304RA.pdf

[7] IDC. *Open Source in Global Software: Market Impact, Disruption, and Business Models.* http://www.idc.com/getdoc.jsp?containerId=202511

[8] *Binary Compatibility Issues With C++.* http://techbase.kde.org/Policies/Binary_Compatibility_Issues_With_C%2B%2B

[9] E. Gamma et. all *Design Patterns: Elements of Reusable Object-Oriented Software*

[10] *Using the GNU Compiler Collection (GCC)*, chapter 8. http://gcc.gnu.org/onlinedocs/gcc/Compatibility.html

# The Automated Analysis of Header Files for Support of the Standardization Process

Eugene Novikov
ISP RAS
joker@ispras.ru

Denis Silakov
ISP RAS
silakov@ispras.ru

## Abstract

*This paper considers the method of the header files automated analysis. The method is intended for the LSB standardization process support. The suggested approach is based on the usage of the cpp preprocessor and gcc compiler high-level representation, their extensions and additional analyzers. The basic work stages of tool implementing the suggested method are introduced. Also tool application for the Qt library header files analysis is considered.*

## 1. Introduction

Now in the world a large number of Linux distributions exists [5]. They are widely applied in various systems. All distributions in their basis have the same components. However, components versions depend on system. Also distribution developers make some specific changes for their systems. Therefore at the binary level Linux distributions are not completely compatible and programs, created for one operating system, sometimes can not be started in another without recompilation from their source code. So application developers, who don't publish source code, should either make many efforts and spend much time for maintenance of compatibility at the binary level or release their products for the limited number of distributions [1]. Because of such complexities there are no fully reliable and valid applications in Linux for some software areas [6].

The similar problem of Unix distributions fragmentation has led to POSIX standard creation [7]. This standard contains requirements for core interfaces (about 1000 functions) and does not describe other interfaces. For example, it does not describe graphical interface functions that are used by the most of modern applications.

Project LSB (Linux Standard Base) [2], [4] is aimed to help developers to create and support portable applications for Linux operating system. In all distributions, supporting the LSB standard, binary files of these applications will be executed equally. In comparison with POSIX, LSB standardizes essentially more interfaces (about 40000 functions). The basis of the whole LSB standardization process is the LSB specification database. From its information it is generated:

- LSB standard text;
- primitive tests;
- environment for LSB-compatible applications development.

To estimate the scale, basic objects of the LSB database and their quantity at the current moment are shown in Table 1.

**Table 1. The basic objects of the LSB database**

| | |
|---|---|
| Libraries | ~60 |
| Header files | ~900 |
| Interfaces | ~47000 |
| Types | ~16000 |
| Macros | ~12000 |
| Classes | ~1600 |

From Table 1 it becomes clear that data volume is large. In addition Linux operating system and applications are constantly developed. So the LSB standard covers more and more libraries. Therefore it is necessary to support the database in up to date state. This is done by updating of stored information and uploading of the new one. Thus today the LSB database population automation problem is very actual.

## 2. The review of existing methods and tools for LSB database population

Data collection can be performed on the basis of following sources:

- libraries binary files (with debugging information and without it);
- libraries header files.

At the moment the LSB database is populated in general with data obtained from the libraries binary files analysis. Considerable data volume is modified and uploaded manually.

There are tools that automate both binary and header files analysis. libtodb and libtodb2 [8] allow to process binary files. These tools are based on the readelf system utility [11]. For header files processing the tool headertodb [8] based on the ctags program [9] may be used. These tools allow to automate the process substantially but they do not give all needed functionality.

The main advantages of the binary files analysis are possibilities to obtain:

- the list of interfaces exported by library;
- binary symbols versions.

This data is important to provide with compatibility at the binary level support. Disadvantages of the libtodb and libtodb2 analysis are:

- absence of some data at the binary level (e.g. inline functions);
- insufficient C++ support at the binary level;
- absence of interrelations between complex types at the binary level;
- complexity of the large data volumes processing by means of libtodb.

The tool headertodb allows to obtain a considerable part of data from header files but, due to insufficient analysis of the ctags program, it does not provide demanded completeness and quality. The C++ analysis also is not supported at the necessary level.

In this paper the header files analysis method that allows to eliminate the most of disadvantages specified above is described. This method makes analysis and population the LSB database LSB with obtained information essentially automated. Also the tool implementing the suggested method is described. It is necessary to notice that many features of the method and the tool are connected with necessity of their application for Qt library header files processing [12]. These header files are written in C++ language. Previously they were processed mainly by hand. This paper considers practical results of these header files automated analysis.

## 3. The method of the header files analysis

To obtain data from header files the cpp preprocessor and the gcc compiler parser [11] are used. The suggested approach has such advantages in comparison with the usage of other programs (e.g. ctags mentioned above) and the creation of the own analyzer as:

- Presence of analyzers (cpp and gcc) which are constantly supported by third-party developers. These analyzers allow to obtain data from the high-level representations that posses considerably more simplified and formalized structure in comparison with initial representation of header files (i.e. a standard text). This was the key feature in choosing of the method to be used.
- Maintenance of the most detailed analysis of Linux libraries header files. It is due to very close connection of the cpp preprocessor and the gcc compiler with the given operating system. Header files are processed by them with specific Linux features and different extensions, many of which are made especially for the given preprocessor and compiler. In addition, the third-party analyzers automatically carry out the code correctness check and connect header files with each other.
- The openness of the cpp and gcc source code. This important feature is necessary for more accurate understanding of their high-level representations structure. Also this allows their modification needed to obtain additional data.

Disadvantages of such approach are:

- Absence of the detailed documentation on the used high-level representations. Syntax and semantics of these representations were explored by testing on different examples. It is important to notice that all the necessary, what may be obtained from these representations for the Qt library header files analysis, was studied and implemented. Necessary researches were performed basically directly on header files from the given library and also on more simple examples, including header files written in C language.
- Incompleteness of data obtained from the high-level representations. First of all this comes from the fact that cpp and gcc are intended to compile programs and do not specialize in these representations generation. Therefore there may be no some needed information in the preprocessor and compiler representations. In that case some cpp and gcc source code modifications, extending these high-level representations, and additional analyzers are made. Extensions and additional analyzers will be introduced below.
- Possible preprocessor and compiler representations changes by third-party developers require changes in their analysis. It is necessary to tell that during the time, while the tool processing these representations was developed (about one year), syntax has not changed at all while small changes in semantics for some entities have facilitated their interrelations analysis.

Thus, it becomes clear that disadvantages of the suggested method may be eliminated and in many respects it was already done. Therefore it is possible to enjoy advantages of the approach. Following sections describe this.

## 3.1. The cpp preprocessor high-level representation structure

For parsing tree generation discussed below the compiler gcc uses header files source code after preprocessing by cpp. So the parsing tree does not contain any information on preprocessor directives. The cpp preprocessor output and the additional analyzer are used for the preprocessor directives analysis. The preprocessor output is used to obtain the lists of:

- macrodefinitions declarations (directive `#define`);
- removals of macrodefinitions declarations (directive `#undef`);
- included header files (directive `#include`).

In this output there are auxiliary strings between which there are directives `#define`, `#undef` and `#include` and preprocessed code of the analyzed header file and included header files. It is important to notice that:

- Comments and non-significant spaces (for example spaces between # and a directive name) are removed from directives.
- Multiline directives are transformed into one-line ones with corresponding additional empty strings.
- Lines numbers in the cpp output and directly in header files coincide with each other. Also each directive is fully placed on its own string. These features are important to order preprocessor directives with other entities from header files.

Here formats of auxiliary strings and directives `#define`, `#undef` and `#include` are described:

- `# a line number in a header file “an absolute path to a header file”`
  - o Subsequent strings, up to the next auxiliary string or to the end of file, are content of the analyzed or included to it header file. This content begins with the given line number.
  - o Absolute paths uniqueness allows to distinguish header files with identical short names, e.g. time.h and sys/time.h. So this gives the way to order entities from different header files. In addition, absolute paths to included header files are remained for their unique identification.
- `# 1 “<built-in>”`
  - o Following strings contain system macrodefinitions that are declared by the preprocessor implicitly.
  - o This list may be obtained by means of the headertodb3 tool described below.
- `# 1 “<command-line>”`
  - o Following strings contain macrodefinitions that are declared by the user in the cpp preprocessor command-line.
  - o User macrodefinitions may be declared by means of the headertodb3 special option. In particular, it is possible to redefine system macrodefinitions.
- `#define a macrodefinition name a macrodefinition value`
  - o Macrodefinition declaration.
  - o A macrodefinition value is arbitrary string, in particular it may be absent.
  - o Functional macrodefinitions are declared in form: `a functional macrodefinition name (parameters names divided by commas) a functional macrodefinition value`.
- `#undef a macrodefinition name`
  - o Macrodefinition declaration removal.
- `#include “a header file name”` or `#include <a header file name>`
  - o This means that a currently processed header file includes another one. Included files are searched according to the standard preprocessor rules. Therefore it is important to specify correct paths to directories that will be explored for included headers.
  - o Following two strings contain absolute paths to including and to included header files.

## 3.2. The gcc compiler high-level representation structure

Preprocessed header files are processed by means of the gcc compiler parser. This parser allows to obtain the parsing tree. The parsing tree is the internal compiler structure that is used to represent information on source code. The tree is generated by gcc after lexical, syntactic and semantic analyses. Also the compiler may write the parsing tree in the text representation. The parsing tree text representation structure is not well documented. Research of this structure on examples allowed to make its description.

The parsing tree consists of nodes and their attributes. Nodes correspond to entities. Attributes describe their kinds and properties. In the text representation all nodes are written at the beginning of strings in the form `@integer`, e.g. `@475`. After a node name a list of attributes corresponding to this node is written. The first attribute specifies an entity kind. For example, to represent the integer type the first attribute `integer_type` is used. The first attribute has a name and has no value. Below various entities kinds will be characterized directly by the first attribute.

Subsequent attributes describe entity properties. In the text representation they have the following form `an attribute name: an attribute value`, e.g. `name: @249`. An attribute value may be one of the following kinds:

- a reference to another tree node (e.g. `@12`);
- some text information (e.g. `long double`);
- a place in a header file where an entity is declared (e.g. `/usr/include/time.h:412:12`).

A reference to another tree node specifies that some entity property is described by means of another entity represented by another node. For example, the function declarations `function_decl` may have attribute `name: @11` that refers to the identifier entity `identifier_node`. In turn the identifier has the attribute with the function name value. The function name is written in the usual text form and belongs to the second kind of attributes. An entity declaration place is described in the following form `an absolute path to a header file: a line number in this file: a column number in this file`. Such form is the gcc compiler extensions. These extensions were made since the original declaration place form looked like `a header file name: a line number in this file`. It appeared that it is not enough both to uniquely identify header files (for example, entities from header files time.h and sys/time.h were concerned as belonging to the same header file time.h) and to order entities placed at the same string. Thanks to compilers extensions these disadvantages were eliminated. To designate system entities `<built-in>` is used as a header file name. System entities are, for example, `intrinsic` types such as `void`, `int`, `bool`, etc. Thus attributes values belong to one of three classes each of which is processed in the appropriate way. Some attributes may have identical names, e.g. `note` and `spec`. For such attributes all their values are obtained as values vectors. Despite attributes names coincidence for some entities all properties are unambiguously and consistently defined by a pair `an attribute name, an attribute value`.

Entities, important to obtain the necessary information from header files, belong to one of the following classes:

- declarations;
- types;
- constants;
- auxiliary entities.

The complete description of all entities structure and the method of their analysis is huge and is not considered within the bounds of this paper. Nevertheless the description of four representatives for each class is given below. For such description the analysis of entities attributes and properties that may be obtained is done.

**3.2.1. Function declaration**. The first attribute of this entity is `function_decl`. Subsequent attributes names and values are:

- `name` is a reference to an identifier `identifier_node`, a function name.
- `type` is a reference to a functional type `function_type`, a complete description of a function signature.
- `scpe` is a reference to a function scope, whether a namespace declaration `namespace_decl` or a parent class `record_type`.
- `srcp` is a function definition place in a header file.
- `note` (optional and multiple-valued) is `member` (for a class method) or `constructor` (a class constructor) or `destructor` (a class destructor) or `operator an operator name`.
- `accs` (optional) is access to a class method. It may be `pub`, `priv` or `prot` that accordingly designates public, private and protected.
- `spec` (optional) is a class method specifier. It may be `virt` or `pure` that is designations for virtual and pure virtual accordingly.
- `args` is a reference to `parm_decl`, the first element of a function arguments list containing their types, names and qualifiers.

To obtain additional needed information on a function declaration the following extended attributes are added:

- `ext_qual` (optional) is a class method qualifier. It may be `const` or `volatile`.
- `ext_note` (optional) is `explicit` (an explicit class constructor), `extern` or `static` (a function specifier), `inline` (an inline function), `throw` (a throw function).
- `ext_body` (optional) is a reference to the beginning of a tree expression corresponding to a function body.
- `ext_body_open_brace` (optional) is an opening brace place, the beginning of a function body.
- `ext_body_close_brace` (optional) is a closing brace place, the end of a function body.

According to attributes following information on a function declaration is obtained:

- A function name or an overloaded operator name.
- Return value type.

- For function arguments their types, names, default values and qualifiers are obtained. If a function arguments list has not variable length, last parameter is always `intrinsic` type `void`.
- A function parent, a namespace or a class.
- Whether a function is extern or static, inline, throw, virtual.
- For methods access to them, whether a method is const or volatile, whether a method is a constructor or a destructor are obtained. For constructors explicit property is obtained.
- An inline function body place in a header file. It is necessary to take corresponding block directly from a header file without expression parsing.

**3.2.2. One-dimensional array type**. The first attribute of this entity is `array_type`. Subsequent attributes names and values are:

- `elts` is a reference to an arbitrary type, a type of one-dimensional array elements.
- `domn` (optional) is a reference to an integer constant `integer_cst`, the number of one-dimensional array elements.

It is important to notice that multidimensional arrays are formed as a one-dimensional arrays sequence. A base types sequence is 'natural' i.e. from the fullest type to the basic one. Therefore processing of multidimensional arrays is recursive. In case of empty array usage (for example, `int []`) `array_type` has not attribute `domn`, containing the number of elements. Also it is necessary to pay attention, that in one-dimensional (multidimensional) array passing to function as argument, it is transformed to a pointer (an array of pointers). For example, `int [10][20][30]` becomes `int * [20][30]`.

**3.2.3. Integer constant**. The first attribute of this entity is `integer_cst`. Subsequent attributes names and values are:

- `type` is a reference to an integer type `integer_type` or to an enumeration type `enumeral_type` that is an integer constant type.
- `low` is some arbitrary integer number, an integer constant value.

Integer constants are enumerations elements values, arrays and bitfields sizes, global variables default values and function arguments or templates parameters default values. In the case when used integer constant belongs to some enumeration type, its name must be used instead of corresponding numerical value.

**3.2.4. One-linked list element**. The first attribute of this entity is `tree_list`. Subsequent attributes names and values are:

- `purp`:
  - o for an enumeration type `enumeral_type` is a reference to an identifier `identifier_node`, an enumeration element name;
  - o for a functional type `function_type` (optional) is a reference to an arbitrary constant, a function argument default value;
  - o for a template declaration `template_decl`, an attribute `inst` value is a reference to `tree_vec`, an attribute `prms` value is a reference to `integer_cst`.
- `valu`:
  - o for an enumeration type `enumeral_type` is a reference to an integer constant `integer_cst`, an enumeration element value;
  - o for a functional type `function_type` is a reference to an arbitrary type, a function argument type;
  - o for a template declaration `template_decl`, an attribute `inst` value is a reference to `record_type` (a template instance), an attribute `prms` value is a reference to `tree_vec`;
- `chan` (optional) is a reference to `tree_list`, a following element of an one-linked list.

One-linked lists are processed depending on a context in which they are used.

### 3.3. The additional analyzers

There are not preprocessor conditional compilation directives in the cpp output. Therefore they are analyzed directly through header files. The following conditional compilation directives groups are processed:

- *Conditions* are preprocessor directives specifying the beginning of conditional compilation. `#if`, `#ifdef` and `#ifndef` belong to this group. Conditional expressions must be for these directives. A conditional expression follows a directive name.
- *Branches* are instructions about possible variants in execution. Directives `#else` and `#elif` belong to this group. Conditional expression must be for directive `#elif`.
- *The end of conditional compilation* is expressed by means of directive `#endif`.

During the conditional compilation directives analysis non-significant spaces and multiline strings are processed. Such analyzer does not possess the cpp preprocessor completeness. Therefore in future some corresponding preprocessor output extensions will be probably made.

Besides, auxiliary comments, having the special form, are processed:

- *LSB parameters* are instructions about how included header files must be analyzed. These comments are placed at the beginning of header files from the beginning of lines. They have such form `/* LSB PARAM: a parameter name */`. At the moment it is necessary to process the following *LSB parameters*:
  - *fresh* means that entities from an included header file should be processed as well as entities of an analyzed header file.
  - *print* means that *LSB IDs* discussed below are printed for entities from included header file.
  - *end* means the end of a *LSB parameters* section.
- *LSB IDs* are unique integers which are associated with entities from included header files. These integers are obtained during the corresponding header files analysis and kept into LSB database. Comments are placed on strings before corresponding entities. They have such form `/* LSB ID: an integer number */`. Then *LSB IDs* are used to refer on already processed entities.

Special comments are written into header files mainly automatically. This is done for files that were already analyzed and generated on the basis of the LSB database. These comments may be located in header files of any acceptable inclusion depth. So they are analyzed for all these levels. The latter is possible because of the preprocessor output has all included header files content. In the future new special comments may occur to provide additional needed functionality.

### 3.4. The high-level representations extensions

To obtain some additional information a number of gcc parser extensions was made. Some extensions were already mentioned above. At the moment the full extensions list is the following:

- More detailed entities definitions place in header files.
- Distinction between structure and class types joined by the compiler together into the general internal type `record_type`.
- Attributes correct form i.e. `an attribute name: an attribute value` (e.g. attribute `bitfield` for bitfields).
- Access to type declarations scoped in a class.
- Function declarations properties (inline, throw, const, volatile, etc.).
- Function bodies trees included in the general parsing tree.
- Function bodies in the form of their places in header files.
- Exception types lists.
- Escape-sequences in string constants.
- Function definitions and type declarations prototypes places.
- Absence of the unnecessary precompiled header files generation.
- Absence of information on the unnecessary gcc system functions in the tree text representation.

Extensions are made on the basis of the gcc compiler source code analysis. Usually they do not demand any considerable changes and do not affect the gcc work. Extensions supplement the compiler high-level representation and give more detailed set of entities properties. It allows to avoid the difficult manual analysis. So it automates populating of LSB database with data in very considerable degree. Some extensions optimize the compiler analyzer work. Therefore in future the extensions list will be increased to perform more and more detailed and qualitative analysis as fast as possible.

## 4. The headertodb3 tool

The tool called headertodb3 was developed to implement the high-level representations analysis. Tool input is libraries header files. A great number of various options allow to control the headertodb3 analysis and output. Also many settings are available through the special configuration file. Headertodb3 works automatically after options specifying. If corresponding options are enabled then headertodb3 informs about its actions and displays additional debugging information. In case of some critical error occurrence (e.g. error in cpp or gcc work) the tool finishes work with the corresponding return value. During the work the tool performs the following steps:

- Command-line options processing. Depending on these options the tool performs different analysis kinds. Below the headertodb3 standard work scheme is presented.
- Initializations that headertodb3 uses in the work. The main is the primitives initialization. Primitives consist of `intrinsic` types, C and C++ keywords and some auxiliary information. Primitives are used during the parsing tree analysis to connect tree conceptions with the language ones.
- The preprocessor directives analysis on the basis of the cpp output.

- The special comments analysis on the basis of the cpp output with comments. It is made by means of the additional analyzer.
- The conditional compilation directives analysis by means of the additional analyzer.
- The gcc compiler parsing tree generation. Its conversion to the tool internal representation that will be used in the further parsing tree analysis.
- The parsing tree analysis. This is the main stage during the tool work. The general work scheme at this stage is the following:
  - entities ordering in that sequence in which they are encountered in header files;
  - the ordered entities analysis by means of the special handlers.
- Temporary folders and files removal.

During its work the tool generates the following files into the special folders:

- The text file with information on preprocessor directives. This information is used by other tools to obtain inclusion interrelations between header files. It is needed to construct a sequence in which header files will be analyzed by headertodb3. By means of the tool special option this analysis may be standalone.
- The major SQL-script with information on all entities, their properties and interrelations. The LSB database may be directly populated with this script. It is necessary to notice that some additional tables are used in the database to store information from C++ header files. Then other tools use information from the LSB database to check different dependencies and to generate header files and corresponding to them HTML pages.
- The text file with information on function bodies places in header files. This may be used for the following functions bodies processing.
- The text file containing information on errors that the tool faced during the work. Information on failures kinds and failures locations is printed here.
- In addition by means of the tool special options it is possible to obtain text and HTML representation of the gcc parsing tree. The parsing tree in the HTML page form is convenient for navigation between nodes. Therefore everybody can walk quickly through the parsing tree by means of a usual browser.

The tool was developed by degrees. The tool first version is headertodb2. It is intended to process header files written in C language. During the headertodb3 creation the corresponding experience was taken into account and many innovations were brought. Tools were tested on real header files from C and C++ libraries. Also the test scenarios were made on the basis of different real situations and arisen errors. They consists of about 1000 various test cases that cover C and C++ languages and the compiler extensions. All these test cases may be passed automatically. It is very important to immediately trace malfunctions that may occur during tools development. Thanks to the automated testing system both headertodb3 and other tools errors were found out.

At the moment the most important check of headertodb3 tool is its application in Qt4 library [12] header files processing. By means of the tool information shown in Table 2 was obtained.

**Table 2. Headertodb3 application in the Qt4 library header files analysis.**

| | libQtCore | libQtGui |
|---|---:|---:|
| Header files | 85 | 183 |
| Interfaces and their parameters | 4450, 4520 | 9720, 10360 |
| Classes, structures, unions and their fields | 440, 385 | 990, 610 |
| Enumerations and their constants | 160, 1975 | 320, 2180 |
| Templates, their parameters, instances and specializations | 240, 340, 130, 80 | 155, 155, 150, 0 |
| Type declarations by means of typedef | 290 | 240 |
| Macrodefinitions | 290 | 190 |
| Included header files | 190 | 460 |
| Entities including auxiliary ones | 22760 | 40310 |
| Properties and interrelations of entities | 87830 | 161030 |

Also headertodb3 was applied in other Qt4 header files processing and in Qt3 library processing.

## 5. Conclusion

At the moment the header files analysis needed for populating the LSB database with data is one of the most important stages of the LSB standardization process. Initially header files written in C++ language were processed either manually or with the usage of the analyzers which do not correspond to all demands. As a result huge human resources were required to solve this problem.

This paper considers the method that allows to automate the C++ header files analysis. The approach is based on the usage of the cpp preprocessor and gcc compiler high-level representations. It allows to use analyzers for header files from third-party developers and to process and extend the more strictly formalized and simpler high-level representations. The suggested method was implemented in the headertodb3 tool.

The tool was used to populate the LSB database with data within the Qt library standardization process. On the basis of tool application results it is possible to make the following general conclusions:

- The method allows to analyze header files with high quality.
- The approach provides required C++ support.
- The tool does not demand considerable computing resources and time expenses.
- The tool makes the header files analysis substantially automated.

In the future additional high-level representations extensions are supposed. It will allow to analyze header files in the more qualitative and automated way.

## References

[1]. A.I.Grinevich, D.A.Markovcev, V.V.Rubanov. Linux systems compatibility problems. The Institute for System Programming proceedings, the 10th volume: “Linux systems reliability and compatibility ensuring”. In Russian.

[2]. A.V.Horoshilov. Linux Standard Base: success history? The Institute for System Programming proceedings, the 10th volume: “Linux systems reliability and compatibility ensuring”. In Russian.

[3]. D.V.Silakov. Current state and perspectives of the LSB infrastructure development. The Institute for System Programming proceedings, the 13th volume, the 1st part. In Russian.

[4]. Linux Standard Base. http://www.linux-foundation.org/en/LSB

[5]. The Linux distributions list. http://www.lwn.net/Distributions

[6]. D.Shurupov. The programs lack as a barrier to the Linux popularization. In Russian. http://www.nixp.ru/articles/plugging_linux_holesD.Shurupov

[7]. IEEE POSIX® Certification Authority. http://standards.ieee.org/regauth/posix

[8]. The LSB Infrastructure Project. http://ispras.linux-foundation.org

[9]. The ctags program. http://www.ctags.sourceforge.net

[10]. The cpp preprocessor and the gcc compiler. http://www.gcc.gnu.org

[11]. The readelf system utility. http://www.opensourcemanuals.org/manual/readelf

[12]. The Qt library. http://www.qtsoftware.com

# The Boost.Build System

Vladimir Prus
Computer Systems Laboratory
Moscow State University, CS department
Moscow, Russia
vladimir.prus@gmail.com

*Abstract*—**Boost.Build is a new build system with unique approach to portability. This paper discusses the underlying requirements, the key design decisions, and the lessons learned during several years of development. We also review other contemporary build systems, and why they fail to meet the same requirements.**

## I. Introduction

For software projects using compiled languages (primarily C and C++), build system is the key element of infrastructure. Mature tools such as GNU Make[1] or GNU Automake[2] exist. However, those tools are relatively low level, and hard to master. They also have limited portability. For that reason, many software project do custom work on build system level. One such project is C++ Boost [3].

C++ Boost is a popular collection of C++ libraries, many of which are either included in the next revision of the C++ Standard or planned for inclusion[4], [5]. This unique position attracts a lot of users, who, in turn, use a wide variety of operating systems and differently-configured environments. This differs from most commercial projects — which target a few platforms that are important from business perspective, and are built in a well-controlled environment. This is also different from most open-source projects — which tend to focus on GNU/Linux environment. Developers' background also considerably differs — a person who is expert in C++ is not necessary an expert in different operating systems.

This diversity in user and developer base lead to the following requirements for a build system:

1) "Write once, build everywhere". If a library builds on one platform, it should be very likely that it builds on all other platforms supported by the build system. It follows that build description should be relatively high-level, and avoid any system- or compiler- specific details, such as file extensions, compiler options or command-line shell syntax details.
2) Extensibility. Adding support for a new compiler or a platform should not require changing build system core or build descriptions for individual libraries. Ideally, user would have to write a new module and provide it to the build system.
3) Multiple variants. The build system may not require that the build properties for the entire project are specified up-front, during special "configure" step, and then require that all build products be removed before changing build properties. Instead, it should be possible to change build properties without full rebuild. Such change may happen at three levels:
   a) Between different parts of the project. The simplest example is compiling a specific source file with an additional compiler option. More complex example is building a specific module as a static library, and another as a shared library. It should be possible to change every aspect of the build process — even including the used compiler.
   b) Between different builds. For example, one may originally build a project in release mode for testing and, after discovering a bug, wish to initiate a build in debug mode. It would be wasteful to remove the previously built object files, so the build system must arrange for debug and release products to be placed in different directories. The mechanism should not be restricted to just debug and release builds, but apply to any build properties.
   c) Within one build. This means that one invocation of the build system may produce several variants of a build product — for example, static and shared versions of the same library.

This paper describes Boost.Build[6] — a build system developed to meet the above requirements. Section 2 will describe key concepts and mechanisms of Boost.Build. In section 3 we review the lessons learned during development as well as some unexpected drawbacks. Section 4 discusses other contemporary build tools, and why they could not be used. Section 5 summarizes the article and suggests future development directions.

## II. Design concepts

The best way to explain the key design elements of Boost.Build is by following a few steps of gradual refinements, starting from a classic tool — GNU Make. In GNU Make, a user directly defines a set of *targets*, where a target is an object that has:

- a name
- a set of *dependency targets*
- a command to build the target

Consider this example:

```
a.o: a.c
  g++ -o a.o -g a.c
```

Here, the name of the target is `a.o`, the only dependency is a target named `a.c`, and the command invokes the `gcc` compiler. Given the set of targets defined in a build description file ("buildfile" for short), GNU Make identifies targets that are out of date with respect to their dependencies, and invokes the commands specified for such targets. The description shown above has two problems. First, the names of the targets and the exact commands typically may vary depending on environment, and should not be hardcoded. This problem is typically solved using variables — in the example below, the OBJEXT and CFLAGS variables may be defined as appropriate for platform.

```
a.$(OBJEXT): a.c
  g++ -o a.o $(CFLAGS) a.c
```

While this makes build description more flexible, it also makes it rather verbose, and hard to write. Second, depending on build variant and platform, even the set of targets may vary. For example, depending on platform and desired linking method, building a library might produce from 1 to 4 files. Obviously, conditionally defining 4 targets for every library is extremely cumbersome.

Modern build systems do not require that user describes concrete targets, but provide a set of *generator functions*, or *generators* for short. A generator is called with the name of primary target, a list of sources, and other parameters, and produces a set of concrete targets. Sometimes, these concrete targets are GNU Make targets. Sometimes, a different low-level build engine is used. For example, a library might be defined like this[1]:

```
library(helper, helper.c)
```

This statement calls a function `library` that constructs the set of concrete targets that is suitable for the target platform — which may include the library file proper, import library, and various manifest files. The compiler and linker options are also derived from both the platform and the way the build system was configured. Thus, the user-written build description does not include any platform-specific details. Instead, such details are handled by the build system, which is separately maintained. Boost.Build also uses a similar description mechanism, but advances it further.

First key observation is that using generators is not sufficient to achieve portability. Requirements listed in section 1 include using different build properties for different parts of the project. This can be achieved using additional parameters to generators. But if the set of those parameters, or their values, is either incomplete, or depends on platform, the build description is not portable. To achieve the portability goals, Boost.Build defines a large set of build parameters with the following characteristics:

- Every generator accepts the same set of build parameters
- The names of build parameters and their values are the same everywhere

[1]For presentation purposes, we have abstracted away the syntax of modern build systems.

For example, every generator in Boost.Build accepts a parameter named `optimization`, with `none`, `speed` and `space` as possible values. Consequently, the example might be modified as follows:

```
library(helper, helper.c,
            optimization=space)
```

This change addresses requirement 1 ("write once, build everywhere").

The second key observation is that differences between platforms are so significant that creating a single generic generator such as `library` is hard. Obviously, small behaviour differences can be handled in an ad-hoc way — for example by introducing a global variable set by platform-specific code and checked by the generator. However, in existing tools there are dozens of such variables, with the generator still containing significant platform specific logic. More systematic approach is needed. To that end, Boost.Build allows several generators to exist, and uses a dispatching function to select the generator to use. In example below:

```
library(helper, helper.c,
            link=shared)
```

the description written by the user looks the same as before. However, the `library` function is no longer responsible for constructing targets. Instead, it merely selects and invokes a platform-specific generator. This generator need only deal with a single platform, and can be easily implemented. The specific generator selection algorithm (that will be described below) allows new generators to be defined in platform-specific modules and automatically participate in generator selection, thereby addressing requirement 2 ("extensibility"). It should be noted that recursive calls are common — for example, the `library` generator might use the `object` generator. In Boost.Build, the dispatching function is also used for such recursive calls, allowing for fine-grained customization.

The third key observation is that if a build description is allowed to call a dispatching function when the build description is parsed, it severely limits the possibilities to further build the same part of a project with different properties. To address this issue Boost.Build introduces *metatargets* — which are essentially closure objects. Consider an example using the actual Boost.Build syntax:

```
lib helper : helper.cpp ;
```

This statement defines a closure of the dispatching function, binding the name and the sources list. If we invoke Boost.Build from the command line using the following command:

```
$ b2 toolset=gcc variant=debug
link=shared
```

then the closure object will be called with the specified build parameters. The `toolset=gcc` and `link=shared` parameters uniquely specify a generator — `gcc.link.dll` — that is called to produce the concrete targets. In the example below, we request a two-variant build:

```
$ b2 toolset=gcc link=shared --
toolset=msvc
```

In this case, the created closure object will be called twice, once with `toolset=gcc` and `link=shared` parameters, and once with `toolset=msvc` parameter. Different generators will be selected, and a substantially different set of concrete targets will be produced. The metatargets mechanism addresses requirement 3 ("multivariant builds").

We have introduced the key design elements of Boost.Build. The remainder of this section describes in detail the most important mechanisms used to implement this design.

## A. Requirements

It is uncommon for the entire project to be buildable for all possible build parameters. *Requirements* is a mechanism to restrict the possible build parameters for a specific metatarget. Simple requirement merely state that a given build parameter should always have a specific value for this metatarget. For example:

```
lib helper : helper.cpp
     : link=static ;
```

overrides the value of the `link` build parameter that was passed to the metatarget, and causes the concrete targets to be constructed as if `link=static` was passed. Conditional requirements override a build parameter if some other parameters have specific values. For example:

```
lib helper : helper.cpp
     : toolset=msvc:link=static ;
```

will override the `link` build parameter only if the `toolset` build parameter has the value of `msvc`. Finally, indirect conditional requirements specify that a user-provided function should be called to adjust build properties.

For convenience, a buildfile may specify *project requirements* that are automatically added to requirements of all metatargets in that buildfile.

## B. Platform support

This section explains two mechanisms that facilitate easy support for new platforms — selection of generators by the dispatching function, and translation of build parameters into properties of concrete targets.

Let's look again at the syntax used to declare a metatarget:

```
lib helper : helper.cpp ;
```

As said before, this creates a closure of the dispatching function, binding target name, list of sources, and — which we did not say before — the metatarget type, in this case `lib`. For the purpose of generator selection, Boost.Build maintains additional information about each generator — the metatarget type, and the set of required build parameters. For a concrete example, consider the following table:

| Generator | Type | Required parameters |
|---|---|---|
| gcc.link.dll | LIB | toolset=gcc |
| gcc.link | EXE | toolset=gcc |
| msvc.link.dll | LIB | toolset=msvc |

When the dispatching function is called, it first selects the generators associated with the metatarget type. In our example, such generators are `gcc.link.dll` and `msvc.link.dll`. Then, required parameters of the selected generators are compared with the build parameters passed to the dispatching function. If any of the required parameters is not present, the generator is not considered. In our example, if `toolset=gcc` is passed to the dispatching function, then `msvc.link.dll` generator is discarded. All the remaining generators are called. If exactly one succeeds in generating targets, then the dispatching function returns. Otherwise, an ambiguity is reported and the build process stops. This selection mechanism allows additional generators to be easily added, without modifying core logic of the build system.

When a specific generator constructs a target, it should establish the exact path and name of the target, as well as the command to build it. All that typically depends on build parameters. Of course, a generator may use arbitrary logic to compute this information, but Boost.Build comes with convenient default behaviour. The target path is constructed using the values of build parameters. For example, a path might be `bin/gcc/debug/`. Some mechanisms are used to make the paths shorter — for example, for a few common parameter the path includes only the values, but not the names. Also, parameters that have default values are not included in path. Target name is constructed from the name of the corresponding metatarget. Boost.Build maintains a table, that is indexed by metatarget type and the value of the `target-os` build parameter, and gives suffix and prefix that should be added to metatarget name. The mechanism to construct updating command is the key to easy definition of new generators, and is illustrated below:

```
actions gcc.link.dll {
    g++ -shared $(OPTIONS)
}
flags gcc.link.dll OPTIONS
    : <profiling>on : -pg ;
```

First, a *command template* is defined — by convention, it has the same name as a generator. Command template may refer to variables, in this case `OPTIONS`. The second statement in the example establishes mapping between build parameters and variables that are replaced in command template. Given these declarations, a generator can create a new target specifying `gcc.link.dll` as command template for that target. All the `flags` statements for this command template are automatically processed. The `flag` statement above requests that if build parameter `profiling` has value `on`, then the `-pg` string be added to the `OPTIONS` variable. After all flag statements are processed, every reference to a variable in the command template is replaced by the variable's value. This mechanism proved to be highly beneficial, because it allows to add support for a new build variable to any generator with a very localized change.

## III. Current state and learned lessons

At this point Boost.Build is a mature tool that can be successfully used in production environment, and has already met its requirements. At the same time, it is being actively developed. This section will describe the main issues that were discovered.

### A. *Metatarget-induced indirection*

In most existing build tools, buildfiles are written in some interpreted language, and are executed at build system startup, calling generators and constructing targets. Boost.Build differs from this model by creating closure objects that are called with proper build parameters at a later point. Furthermore, Boost.Build does not require that project be "configured", with some of the build parameters fixed, before starting a build. Consequently, when a buildfile is executed it does not make sense to talk about "current" build parameters and no logic that depends on build parameters may be implemented as an `if`-statement on the top level of a buildfile. Rather, such logic must be implemented as functions that will be called by generators.

For many users, this trait cause understanding problems. We believe that this complexity directly follows from the requirements and key design elements, and cannot be fully eliminated. On the other hand, most users successfully adjusted to this model.

### B. *Code-level extension mechanisms*

One of the goals of Boost.Build was simple description language. This lead to invention of concise syntax for many tasks. However, often no suitable programmatic interface was designed for the case when the concise syntax is not enough. In other words, there are many areas where build behaviour needs to be customized by the user, and there's a wide spectrum of possible customization mechanisms — from a new build parameter to a new metatarget type. In a few cases, this spectrum is not evenly covered, and user has to choose between a very simple method that is not flexible enough, and an extremely complex solution.

One example is the conditional requirements syntax shown previously: `toolset=msvc:link=static`. This syntax is sufficient for the majority of cases, but does not support complex conditions — in fact, conditions using any logical operators except for "and". Until indirect conditional requirements were introduced relatively late during development, users were forced to use a fairly verbose mechanism instead.

Another example is generators. It is very easy for user to declare a new generator that produces one output target from one input target. However, any conditional logic — such as creating an additional target depending on some build parameters — requires substantial complexity. While we have described generators as functions in this article, they are actually implemented as classes in certain programming language, which adds some overhead for just declaring a new generator. Furthermore, the implementation of the base generator classes was not designed for easy extensibility, so often, user had to reimplement significant amount of code.

We believe that issues of this kind have only small correlation with the key design choices, and can be eliminated. In fact, quite a few were already fixed as user report them.

### C. *User expectations*

One unexpected issue during development was users' expectations. It is safe to say that most users either have GNU Automake background, or are not experienced with command line tools, and these users have some specific, and often different, expectations.

For example, GNU Automake allows to change compiler by setting environment variables, such as `GXX`. Users often try the same with Boost.Build, and find that it has no effect. For another example, Boost.Build does not stop after a compile error, but builds other targets that do no depend on the failed one. At the end of the build, a summary of failures is printed. This small change proved problematic. Many users did not understand that the error was printed earlier, and interpreted the summary as the original error. And on some operating systems, finding an error in several thousand lines of build output is a problem itself. Developers on Microsoft Windows operating system usually expect that every tool checks system registry for all configuration. Consequently, they found it very unnatural when prior versions of Boost.Build required to specify compiler location in a configuration file. Finally, Boost.Build command line syntax is slightly unusual, having separate syntax for command line options, build parameters and arguments. Many users still try to use option's syntax to specify build parameters.

Some of those issues are natural consequences of a different design and require users to adjust. But still, many issues are independent, and can be easily addressed. We recommend that design process for any project in an established area include explicit gathering of user expectation to avoid unnecessary differences in operation details.

## IV. Existing solutions

There are two build systems that are most commonly used today – the one integrated with Microsoft Visual Studio, and the Automake build system. However, neither of them is truly portable. Below, we review a few solutions that work across different platforms.

### A. *Eclipse CDT*

The C/C++ Development Tooling (CDT) for the Eclipse Platform comes with its own build system[7]. The CDT build system keeps a repository of available tools, organized in named toolsets. For each tool, input and output file types are specified. Any project is required to specify the name and type of the target that should ultimately be built, and CDT automatically picks tools that can produce the desired final target from all files in the source folders. Each tool can have a set of options that are editable via user interface. Similar to Boost.Build, tool options may be specified on individual source folders or individual source files.

Let's review how CDT build system can support the build system requirements described in sectionI:

- The "write once, build everywhere" requirements is not met. User can count on some functionality to be available everywhere — in particular shared and static libraries and predefined "debug" and "release" build variants. However, any further fine-tuning is done via options that are specific to each tool. Therefore, if building with a different compiler, the options has to be specified anew. CDT actually has an indirection level between the value of an option as displayed in user interface, and the command line flags used for compilation when that value is selected. Therefore, it would have been possible to implement a portable set of options, that every tools would translate into appropriate command line options. However, since such portable set is not defined, build descriptions in CDT are not portable.

- The extensibility requirement is poorly met. The only way to extend build system is via new tool definition, and it not possible to completely override the build process for a specific platform. Further, tool definition should be be included either in CDT core or in a separate Eclipse plugin, and cannot be easily packaged with a project. Tool definition uses an XML-based language, which appears to be inconvenient in practice.
- The multivariant requirement is partially met. CDT supports build configurations that include a complete set of options for all tools, and allows one to freely change build configuration every time a build is initialized. Each part of a complex project may use different build configuration. Note that this happens because a "project" in CDT terminology may only contain a single final target, so non-trivial user project has to be split into multiple CDT projects. It is not possible to build the same target in different configurations simultaneously. It is also not possible to build using arbitrary ad-hoc set of build parameters — one has to define a new configuration instead.

CDT has one unique feature — it has a mechanism to specify dependencies between build parameters. For example, 64-bit compilation can be enabled only if the chosen processor indeed supports 64-bit instructions. This feature can be worthwhile to implement in Boost.Build.

## *B. CMake*

The CMake [8], [9] build tool is designed as a abstraction on top of existing "native" build systems. When invoked, CMake reads its buildfiles, and then generators secondary buildfiles using a selected "backend" build system – for example, GNU make. The project is then build when the user explicitly runs the secondary build system. Whenever CMake buildfiles are modified, or any build properties must change, the secondary buildfiles are regenerated. Such scheme improves build system speed for the case where a project is repeatedly build with the same settings.

Let's review how CMake can support the build system requirements described in sectionI:

- The "write once, build everywhere" requirement is not met. The points we have raised when discussing CDT equally apply to CMake.
- The extensibility requirement is not met. Platform-specific modules in CMake essentially specify variables used by CMake core, and it is not possible to completely replace generators[2]. Support for some platforms, for example, Microsoft Visual Studio, relies on special core functionality.
- The multivariant requirement is not met. CMake requires that a project is configured with specific set of properties, and requires reconfiguration for any change in properties. This appears to be result of an explicit design goal, meant to make it impossible to accidentally mix modules built with incompatible settings.

## *C. SCons*

The SCons build tool[10], [11] is unique in two aspects.

First, it uses the Python language for buildfiles, as well as implementation language. In contrast, both Boost.Build and CMake use custom languages. On one hand, this means that the syntax is not as concise, due to punctuation and quoting rules of Python. On the other hand, the use of a mature and widely known programming language reduces learning curve, and simplifies many programming tasks. It also means there's no language boundary between buildfiles and build system core.

Second, SCons uses cryptographic signatures to detect if a target should be rebuild. When a target is built, signature of content of all source files, as well as the command used to produce the file is stored. These signatures are recomputed on every build, and if they differ from the scored ones, the target is rebuild. This approach means that a change in command associated with a target will be detected, and cause a rebuild, while other build system can build different parts of project with incompatible commands.

For the simple usage, SCons provides a set of generators that can be called from buildfiles, for example `Program` and `Library`. The targets produced by those generators use globally specified build parameters. However, SCons also provides a mechanism called *construction environment* — explicitly created collection of build parameters. It is possible to invoke a generator in specific environment, consequently building different parts of projects with completely unrelated set of build properties. However, this mechanism does not allow to easily build one target with different properties. SCons does not assign different directories for target with different properties, so user is required to explicitly specify different names for final and intermediate targets.

Let's review how SCons supports the requirements described in sectionI:

- The "Write once, build everywhere" requirement is not met. The points we have raised when discussing CDT equally apply to SCons.
- The extensibility requirements is met. A platform specific code can completely replace standard generators if so desired.

[2]The "generator" is used in the sense defined in this article. CMake documentation uses the word "generator" for an unrelated concept.

- The build variants requirements is partially met. Similar to CDT, SCons allow to explicitly define several build variants, and unlike CDT, all variants can be built simultaneously. However, SCons does not automatically place products for different parameters in different directories, so it is in general not possible to change any parameter between two build invocation without discarding previous build products.

## V. Conclusions and Future Work

This paper has presented the requirements for the Boost.Build system and its key design decision, as well as reviewed some existing solutions. We believe that the main distinguishing characteristics of Boost.Build are:

- portable build properties, and associated mechanisms like requirements
- true multivariant builds, specifically the metatarget concept
- convenient extensibility, in the form of generator selection and flags mechanisms

While some design complexities were encountered, we believe that overall, Boost.Build is a step forward in the area of software construction.

There are two key areas of future development:

- Use of the Python language for implementation and build description. We find that Python has become sufficiently popular and well-supported and the benefits of using it will outweight slightly more verbose syntax.
- IDE integration. Because Boost.Build does not rely on any legacy backend build tools, and because every metatarget can be repeatedly constructed with different, or same, build properties, it is particularly suitable for integrated development environments — making it possible to quickly determine what products must be rebuild as result of changes made by a user.

## VI. Acknowledgments

The author would like to thank Alexander Okhotin and Sohail Somani for reviewing drafts of this paper.

## References

[1] R. M. Stallman, R. McGrath, and P. D. Smith, *GNU Make*. GNU Press, 2004.
[2] G. V. Vaughn, B. Ellison, T. Tromey, and I. L. Taylor, *GNU Autoconf, Automake, and Libtool*. Sams, 2000.
[3] "C++ boost libraries," http://boost.org.
[4] *The C++ Standard (ISO/IEC 14882:2002)*. Wiley, 2003.
[5] "Technical report on c++ standard library extensions," iSO/IEC PDTR 19768.
[6] "Boost.build website," http://boost.org/boost-build2.
[7] C. Recoskie and L. Treggiari, "Extending the eclipse cdt managed build system," *Dr. Dobb's Journal*, 2007. [Online]. Available: http://www.ddj.com/cpp/197002115
[8] K. Martin and B. Hoffman, *Mastering Cmake*. Kitware, Inc., 2006.
[9] "Cmake website," http://cmake.org.
[10] "Scons website," http://scons.org.
[11] S. Knight, "Building software with scons," *Computing in Science & Engineering*, vol. 7, pp. 79–88, 2005.

# Verification and testing automation of UML projects

Nikita Voinov, Vsevolod Kotlyarov
Saint-Petersburg State Polytechnic University, Saint-Petersburg, Russia
voinov@ics2.ecd.spbstu.ru, vpk@ics2.ecd.spbstu.ru

*Abstract* – **This paper presents an integrated approach to verification and testing automation of UML projects. It consists of automatic model creation from UML specifications in the formal language of basic protocols, model's verification by the means of VRS technology and automatic tests generation in TTCN language using TAT. The actuality of this task arises from necessity of software functionality's correctness checking, including verification and testing, but there is lack of industrial technologies which allow integrating these two activities. Results of the developed approach piloting are also described.**

## I. INTRODUCTION

Documents describing software requirements can contain a large amount of miscellaneous types of errors, the main of which is difference between requirements adopted and detailed by technical specialist and customer's initial requirements. Enormous help in solving this problem is brought by using of formal languages and notations for requirements development. These notations allow customer and developer programmer to speak one language and lead to extermination of specifications' ambiguity of different types. Formal notations and formal methods of software quality control are often used in software development practices. The most popular graphical formal language is UML (Unified Modeling Language) [1]. This standard contains a wide range of graphical objects and allows creating system description from different points of view.

One of the main tasks the software developers have to solve is software functionality's correctness checking, which starts from development of software requirements and lasts until software withdrawal. This causes high demands to completeness and productivity of this checking and leads to appearance of new technologies and program instruments for automation of software functionality's correctness checking, which includes verification and testing.

Although there are a lot of miscellaneous instruments of verification (Spin (BellLabs laboratory), SCR (NavalResearch laboratory), VRS (ISS organization), etc. (see [2, 3] for review)) and testing automation (Rational Rose (IBM), TAT (Motorola), Together (Borland), etc. (see [4] for review)), two serious problems can be stated. The first one is lack of industrial technologies which allow integrating testing and verification. This is especially important when a huge amount of system's behavioral scenarios have to be verified in order to guarantee its correctness and this have to be done in limited time. Secondly, verification based on model checking (when a model of the system is created and requirements for every possible model's state are checked) uses some formal language to create a model of the system and the process of model creation from formal specifications is quite long and laborious.

This paper outlines the main principles of verification and testing automation of UML projects, including automatic model creation from UML specifications in the formal language of basic protocols, model's verification by the means of VRS technology and automatic tests generation in TTCN [5] language using TAT (Test Automation Toolset) [6]. Results of the approach's piloting on large telecommunication program project are also presented.

## II. APPLIED TECHNOLOGIES

### *A. Basic Protocols*

Basic protocol is a formal representation of an assertion about some actions that have to be applied in a program or algorithm under some conditions. In a general case a basic protocol is a Hoare's triplet [7] in the following notation:

$$\alpha \xrightarrow{\mu} \beta$$

where $\alpha$ and $\beta$ are the pre-condition and the post-condition respectively and μ – is the process part of the basic protocol. Both $\alpha$ and $\beta$ conditions are specified by logic formulas of the basic protocols language (a variant of the first-order logic) which can be evaluated for any state of the system.

Basic protocols can be consistently concatenated through their pre- and post-conditions – if the state specified by the post-condition of one basic protocol is to the same as the one specified by the pre-condition of the next basic protocol (actually, the pre-condition formula of the successor should be derivable from the post-condition formula of the predecessor). All such possible concatenations construct the model's behavior graph to be processed by verifier.

The language of basic protocols has an MSC-type syntax [8] and basic protocols can be presented in two ways: textual and graphical (MSC/PR and MSC/GR) [9].

### *B. VRS Technology*

This technology is capable to verify models represented with basic protocols, from small to huge ones. As a result, various incidents of non-deterministic behavior, unreachability of specified system states, or deadlocks are detected. If no such defects are found, the system model is formally proved to be complete and consistent within the specified constraints.

Automated verification of software systems with VRS technology implies the functional requirements, which were used for system implementation, and system's model in the form of basic protocols created from the source code, formal specifications, etc. The technology checks that the model meets the system requirements. This means that the software system satisfies them as well.

For verification process an ordered list of signals or basic protocols that contain required events (actions, signals, etc.) should be specified. VRS can check that for this model the behavior graph contains paths which include the specified sequences in the specified order. The existence of such paths (traces) is a proof of correctness of the model behavior with respect to this criterion. Search of such traces is realized by looking for respective signal interaction between agents or by looking for the specified basic protocol names in the generated traces and considering their actual ordering.

Thus, a trace is a scenario of a possible model behavior. Since the model was derived from an actual implementation of a program system, we can say, that a trace is a scenario of an actual system behavior. Scenarios are represented as consistent concatenations of relevant basic protocols into one chain. VRS outputs traces in the MSC/PR view.

Results of verification are automatically summarized in a verification report, which describes all found inconsistencies, discrepancies, deadlocks, and other errors in the model. Traces demonstrating the incorrect model behaviors are attached to the report. They are used to identify the root causes of such incidents.

Traces generated by VRS can be used for automated creation of an exhaustive test suite for the program system. The TAT (Test Automation Toolset) tool is used for automated test generation from those traces along with the respective testing environment and subsequent test runs.

### *C. TAT (TEST AUTOMATION TOOLSET)*

A key to make testing technologies cheaper and more efficient lies in the area of test generation techniques, i.e. efficient and compact description of test sets and thus significantly reduces tester's manual efforts to develop them. Another key is visualization of formal description by means of graphs. These problems are solved by testing automation tool – test generator TAT.

TAT is a joint toolset, which provides complete, fully automated testing cycle based on user-defined scenarios developed in formal language MSC - Message Sequence Charts.

TAT encompasses several tools that offer complete set of solutions for efficient specification analysis and test generation.

In addition to standardized MSCs, TAT supports extended MSC notations, enhanced with macros, allowing significant reduction of code size developed manually. This extended notation allows absolute, relative, and more complex time specifications in test scenarios. With such framework, TAT helps to get significant time and cost savings through reuse and efficient workaround of time and macro definitions.

## III. MAIN STEPS OF VERIFICATION AND TESTING AUTOMATION

### *A. Autoformalization of UML Specifications*

Before using VRS for verification purposes, system's requirements shall be described in the language of basic protocols. Manual creation of such description is laborious process and can be compared with manual development of test cases in this regard. In some situations the process of formalization can be hastened by automatic creation of basic protocols from initial specifications. This approach is called "autoformalization".

In those UML projects, where system's specification is presented as a state machine, respective set of basic protocols can be generated automatically by using special tool – uml2bp, which in fact is a module of VRS. This module generates sets of basic protocols for every state machine in Telelogic Tau G2 project with .ttp extension as well as creates files with environment and events description, needed for VRS project. All generated files can be imported to VRS for verification.

Example of generated basic protocol is shown in Fig.1.

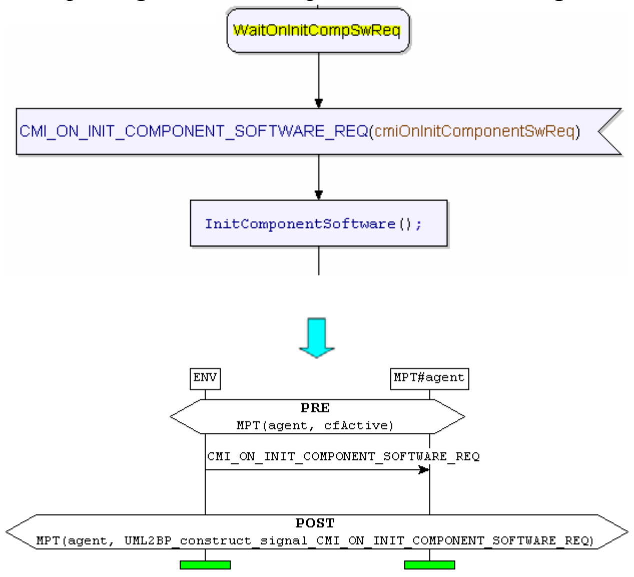

WaitOnInitCompSwReq
CMI_ON_INIT_COMPONENT_SOFTWARE_REQ(cmiOnInitComponentSwReq)
InitComponentSoftware();
ENV
MPT#agent
PRE
MPT(agent, cfActive)
CMI_ON_INIT_COMPONENT_SOFTWARE_REQ
POST
MPT(agent, UML2BP_construct_signal_CMI_ON_INIT_COMPONENT_SOFTWARE_REQ)

Fig. 1. Converting of a piece of UML state machine diagram into a basic protocol

### *B. Vefification with VRS*

#### *Basic Approach to Verification with VRS*

The procedure to check a requirement – is a direct formulation of a sequence of observable causes and results of some activities; after analyzing this sequence a conclusion can be derived whether the requirement is satisfied or not. Such procedure may be used as a criterion for meeting this particular requirement. Basically, the criterion procedure is a method for checking satisfiability of a requirement. Term "chain" can be used for the criterion procedure.

After identifying in the behavioral scenario (hypothetical one or implemented in a real system or system model) the fact that such criterion is satisfied, one can state that the respective requirement in the system being analyzed is satisfied as well.

A procedure for checking requirements (a chain) is specified through formulating all its elements: initial conditions (causes) required for performing a certain activity, the activity itself and observed results of performing the respective activity.

In particular cases to describe the causes and results the states of variables (in form of their values or constraints for tolerance range) may be used. These variables are employed by the activity to track the state changes. In case of non-determinism, possible variants of state changes are tracked. A direct transition from one state to another with a void activity is also possible.

All the above refers to constructing a use-case for a particular requirement. So, a chain or use-case with sequences of activities and states may serve as a criterion for satisfiability of a requirement, sufficient to demonstrate it. Non-determinism in formulations is also covered with a number of chains or use-cases.

*Realization of verification stage*

Step 1. Formulation of filters and heuristics for the current project (after set of basic protocols as well as files with environment and events descriptions have been generated). They help to decrease number of traces by pointing the trace generator to the definite direction.

Step 2. Performing an automatic trace generation cycle.

Step 3. Analysis of findings with deadlocks, inconsistency and other issues, which prevent the tool from completing trace generation in the Goal or Restricted states. Fixing problems identified in findings and their review with the developers. Based on the review results correcting the generated basic protocols or initial requirements. Repeat steps 1-3.

Step 4. Analysis of generated traces with a script to check whether the coverage criteria are satisfied. Repeat steps 1-4 if needed.

Among traces generated by VRS user can choose those which should be used for automatic tests generation with TAT.

*Manual creation of VRS traces*

Another useful feature supported by VRS is manual creation of traces with user defined order of basic protocols (on each step of trace generation user himself chooses the protocol from the list of protocols, which can be applied now). This helps to cover concrete scenario of model's behavior on all possible levels of abstraction. Described below is example of this feature.

Sample UML state machine diagram is shown in Fig.2.

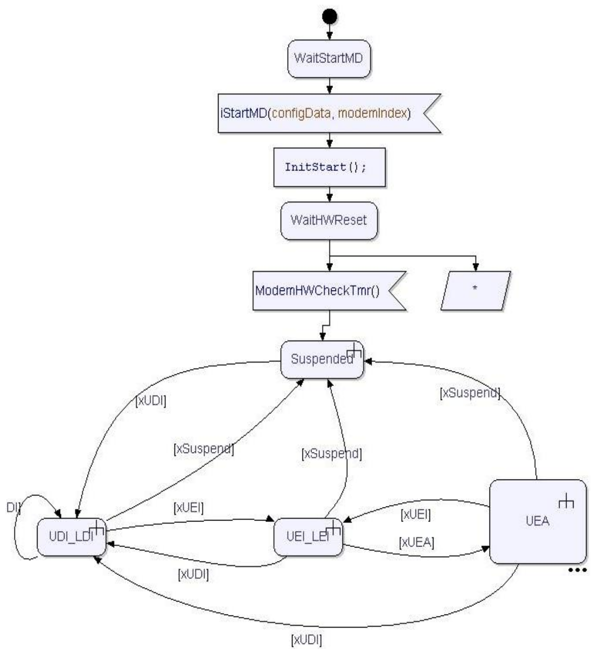


Fig. 2. Sample UML state machine diagram

Its main part is four composite states (Suspended, UDI_LDI, UEI_LEI, UEA) and transitions between them. Each composite state is in turn a state machine itself with complicated behavior inside.

Uml2bp module converts the whole state machine into the set of basic protocols. They are imported to VRS for model's analysis.

Basing on the set of basic protocols, a graph of the model can be constructed. Fig.3 shows the graph without detailed behavior of four composite states.

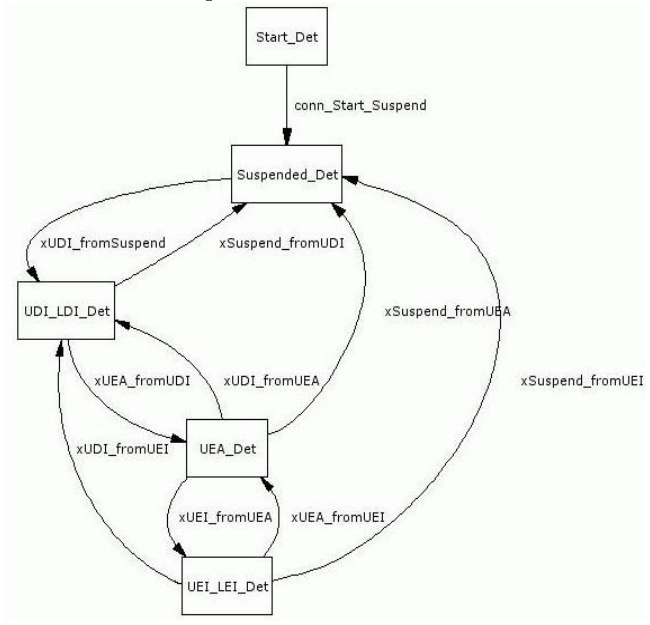


Fig. 3. Graph with composite states

It is also possible to expand any composite state to examine its detailed behavior. Fig.4 shows the same graph but with detailed behavior of Suspended state.

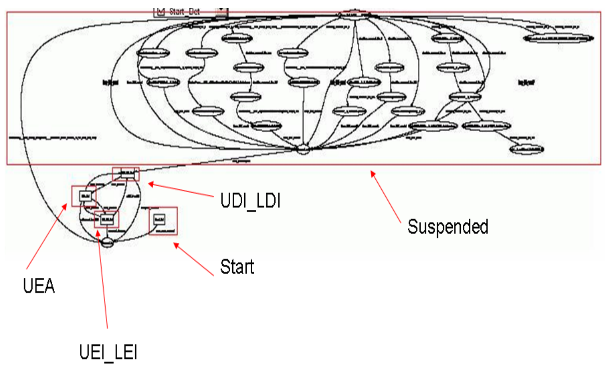


Fig. 4. Graph with detailed behavior of Suspended state

Detailed behavior is represented by a number of states (ovals) with transitions between them. Each transition is performed by a basic protocol. So, in accordance with the graph, one can construct his own trace in interactive mode of trace generation with any abstraction level (high level or detailed) for all composite states and for the whole initial state machine. Traces covering high level behavior of initial state machine and one of possible scenarios of Suspended state's detailed behavior are presented in Fig.5 and Fig.6 respectively.

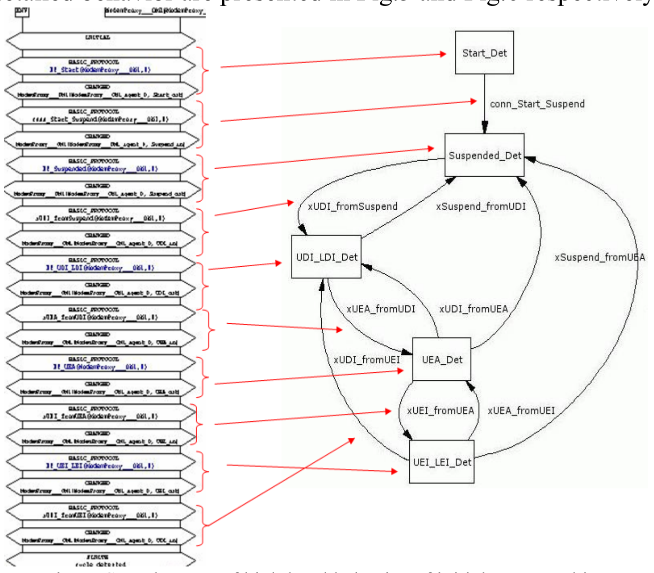


Fig. 5. Sample trace of high level behavior of initial state machine

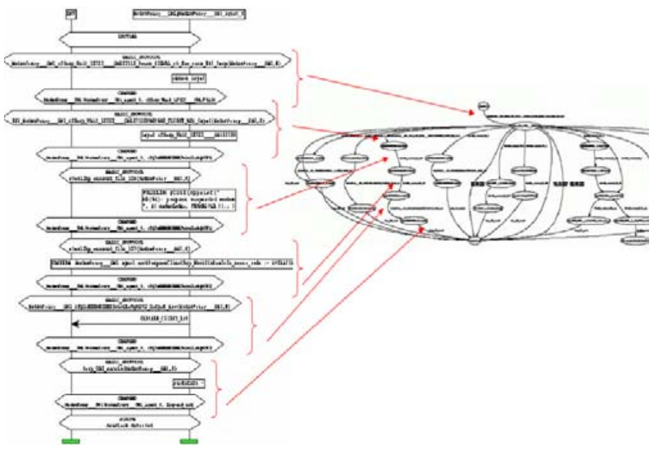


Fig. 6. Sample trace of Suspended state behavior

Now these two traces can be merged: the second trace can be glued in to the respective part of the first one, where Suspended state is covered as composite state. Traces merging is presented in Fig.7.

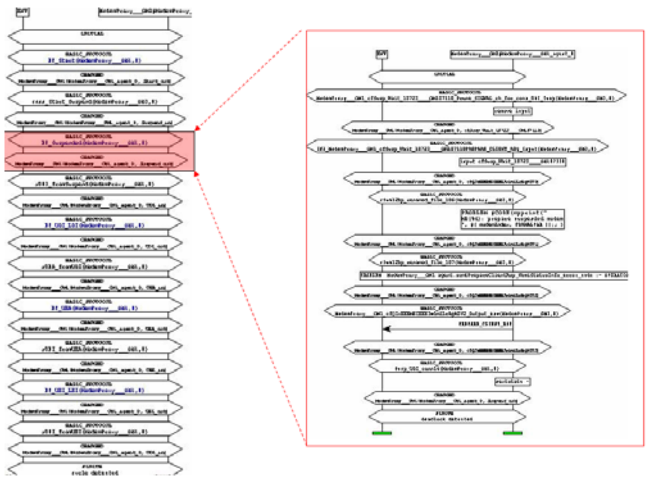


Fig. 7. Traces merging

Traces created in interactive mode can also be used for automatic tests generation.

## C. *Automatic Tests Generation with TAT*

The approach described below is aimed at automatic tests generation on standard language of telecommunication applications testing – TTCN (Testing and Test Control Notation). The proposed approach allows test engineers to exclude manual development and focus on test scenarios, which hastens testing and bugs detection process.

Tests generation process is supported by number of scripts and templates of automatic generation of result TTCN-files. Scripts and templates use auxiliary files with data types description, signals templates description, configuration description, etc.

Tests generation is based on using two input files: a trace (scenario) with signals and their parameters and .xls file, which contains values for parameters used in this trace.

Overall scheme of the process is shown in Fig.8.

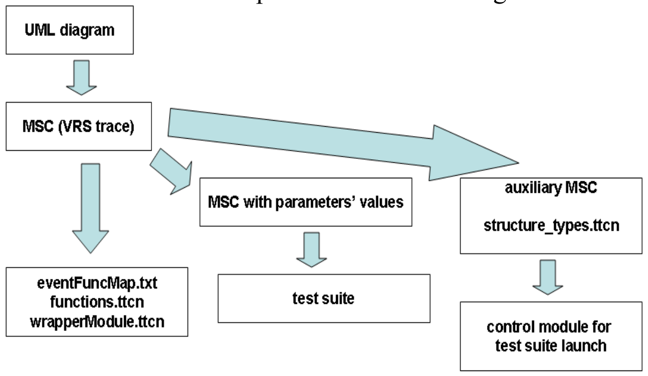


Fig. 8. Automatic tests generation scheme

Several steps can be listed in generation process:

- Automatic generation of functions description, which provide access to the system under test according to signals in the diagram. File with functions description is imported to test project. As a result, only functions calls will be used in test scenario without their bodies.

- Automatic assignment of values to signals parameters in the diagram. Values are taken from .xls file. If one signal is presented several times in the diagram and values of its parameters change, it also should be mentioned in .xls file.
- Generation of test suite from MSC file with assigned parameters values. TAT's template is used on this step.
- Generation of auxiliary TTCN-file, which performs test suite execution.

After that all generated files are imported to test project. As project is compiled and built, test suite can be executed in automatic mode. Test results are saved in log-file.

## IV. RESULTS OF PILOTING

The described approach to verification and testing automation was performed on one of the modules of telecommunication project of wireless network. Estimated size of the whole network's model is about 50000 basic protocols.

About 4000 basic protocols were generated on the stage of autoformalization of the module under test, which took three minutes of uml2bp work. Concerning the fact that estimation for manual creation of basic protocols is 10-50 per day, time saving is obvious and considerable.

Model's verification with VRS discovered about 100 findings. 12 of them were considered to be defects and fixed in future versions of the product.

Automatic tests generation also takes just several minutes, which is much less than manual development. Even considering the efforts required on creation of UML diagram and .xls file for concrete test run adjustment, time savings are estimated as several hours. Besides, when initial requirements are changed or new ones are added, it is enough just to modify initial UML diagrams and spend several minutes on another cycle of tests generation instead of long process of manual correcting the test code with possibility to introduce new errors. Requirements changes during the project realization happen very often, which is caused mainly by large size of projects. This makes the described feature more actual.

## V. CONCLUSION

The developed approach to verification and testing automation proved its advantages in a large telecommunication project and can be further reused in other projects based on UML specifications. As this development process is one of the most preferable nowadays, the field of this approach application enlarges.

## REFERENCES

[1] UML Distilled Second Edition. A Brief Guide to the Standard Object Modeling Language.

[2] Visser W., Havelund K., Brat G., Park S., and Lerda F. Model checking programs. Automated Software Engineering Journal, 10(2), April 2003.

[3] Fernandez J.-C., Jard C., Jeron Th., and Viho C. Using on-the-fly verification techniques for the generation of test suites. In Proc. 8th Conference on Computer Aided Verification, volume 1102 of Lecture Notes in Computer Science, New Brunswick, August 1996.

[4] Drobintsev P.D. Integrirovannaia tehnologia obespechenia kachestva programmnih produktov s pomoshiu verifikacii i testirovania. Kand. dis., SPbGPU. 2006. 238 p.

[5] .ITU-T Recommendations Z.140-142 (2002): The Testing and Test Control Notation Version 3 (TTCN-3).

[6] TAT+Beta User's Manual © 2001-2005 MOTOROLA.

[7] Hoare C.A.R. Communicating sequential processes, Prentice Hall, London, 1985.

[8] Letichevsky A., Kapitonova J., Letichevsky Jr., A., Volkov V., Baranov S., Weigert T. Basic protocols, message sequence charts, and the verification of requirements specifications, Computer Networks: The International Journal of Computer and Telecommunications Networking, v.49 n.5, p.661-675, 5 December 2005.

[9] ITU Recommendation Z.120. Message Sequence Charts (MSC), 11/99.

# Constraint-based Optimizations of Executable UML Models

Andrey Karaulov, Alexander Strabykin

*Abstract* — **this paper describes an ongoing research aimed on the creation and implementation of a set of transformations of executable UML (Unified Modeling Language) models that would improve the execution performance while preserving the behavior. Additional information useful for transformations can be extracted from the constraints embedded in the model. The article contains an example of informal description of a transformation, a scheme of modeling environment extension implementing such transformations, and a review of related tools.**

*Index Terms* — **Constraint Programming, Computer Aided Software Engineering, Model Transformation, Object Constraint Language, Program Optimization, System Modeling, Unified Modeling Language.**

## I. Introduction

IN recent years Model Driven Development methodology (MDD) has drawn much attention among software development industry. According to the MDD software models become the only first class artifacts of the development process; and the whole system creation is seen as a sequence of model refinements starting with the very abstract system model and ending with the model that can be executed performing the functions of the system. This makes models fully reflect the complexity of the system being created and hence readdresses to models many problems previously related to source code in traditional programming languages like Java or C++. Moreover since models contain the description of system behavior precise enough to be executed, the efficiency of behavior specification becomes crucial and cannot be ignored for example for systems that have requirements on their performance. According to our experience complex models usually have a great potential for optimization, however such optimizations are time consuming and error-prone if done manually. Therefore there is a strong need for model optimizing tools.

This paper describes optimization of models in the Unified Modeling Language (UML) [1], since it is de facto an industry standard language for software modeling. However the future results of our study may be applicable to other modeling languages, e.g. domain specific, if they use similar formalisms for specifying system behavior. Behavioral features of the system can be expressed in UML by means of activity, state machine, and sequence diagrams. The latter rather describe a behavior resulted from the interaction of all the participants, but do not exactly specify the behavior of each single party and are used more for scenario definition and logging. Often the expressiveness of UML often is not enough to fully describe the semantic details of a system. In such situations constraints written in Object Constraint Language (OCL) [2] can be used in order to better reflect the semantics. Also OCL can be used to define queries, derived attributes etc, but constraints are usually specified by means of class invariants and pre- and post-conditions of operations.

The goal of our research is creation and implementation of the new methods of optimizing executable UML models. The distinguishing feature of selected approach to model optimization is extraction of additional information needed for model transformations by analyzing constraints embedded in the model. The model therefore is considered as for sure holding all the constraints it contains and the issues of constraint violations are left out of the scope of the paper. This is a common situation when constraints express hardware limitations or other conditions the system should only perform under.

## II. Related Works

A typical model execution scenario consists of several stages including generation of code in a programming language, code compilation and execution. On each stage some optimizing transformations can be applied. For example, compilers that transform source code into executables can apply a variety of optimizations like function inlining, loop unrolling, etc [3]. All these transformations certainly can be applied when a code generated from a model is compiled. Despite most of such transformations can also be applied directly to UML models, since the language standard provides a way to specify all standard actions like loops, conditions etc; in this paper we deal with the higher level transformations that operate on UML models. This approach gives an optimizer the advantage of viewing the system as a whole, since the models fully describe the system behavior. Moreover on a UML level even semantic details expressed in constraints, which are usually unavailable on lower levels, can be analyzed while performing optimization. Optimizations applied to source code in traditional programming languages can have two usually conflicting goals: optimizing either memory usage or performance. Since UML standard lacks information related to memory allocation and distribution our research concentrates

Andrey Karaulov, Alexander Strabykin, Institute for System Programming (ISP) of the Russian Academy of Sciences, Russia, 109004, Moscow, Solzhenitsyna st., 25. Emails: {aka, alexs}@ispras.ru.

on performance optimizations while memory issues are left to be dealt with on lower levels.

Software refactoring is a process of changing the internal structure of an object-oriented program that preserves the observed behavior and is aimed on simplification of modifications and improvement of the readability and design [4]. Since UML is an object-oriented language a lot of refactoring transformations can be applied to UML models [6]; sometimes as a side effect of their application the performance can be improved, but this is not the goal of refactoring. UML models have some specific constructs like state machines and activities that have no direct analogues in traditional object-oriented programming languages. The optimizations of our interest have to preserve the behavior of the system being transformed like refactorings; however the main purpose is a more effective execution of the model. Moreover unlike compiler optimizations optimizing transformations being studied should be visible to user and may rely on user decisions.

UML state machines are based on finite automata formalism, which has been proposed for more than thirty years ago. There are a number of techniques created to minimize automata by means of removing equivalent states [9]. Application of these methods to UML state machines is complicated by the fact that transition equivalence cannot be proved without proving the equivalence of the actions being performed when transition is fired. However proving actions equivalence is not a trivial task if complicated semantics of UML actions is taken into account, since general problem of equivalence of two programs is algorithmically unsolvable.

Another group of related works we are aware of studies the problem of model transformations in general, since it is the key activity of MDD. UML models are usually considered as labeled multigraphs as in [6]. In this case transformation of UML models can be based on graph rewriting formalism. A model optimizer of our interest can be implemented as an extension of a model transformation framework. The most important features of such framework would be extensibility, capability of defining complex and parameterized transformations, and support of constraint analysis in models being transformed.

GReAT language described in [5] is a Graph Rewriting and Transformation language. It contains sublanguages for specifying patterns, transformation rules and control flow for advanced transformation. Extensibility of transformations is achieved by adding user-defined code in a procedural language to specify attribute mapping, which is performed after all graph related operations are done. This approach does not seem suitable for implementing heavy model analysis that must be done before optimizing transformations are executed.

Visual Modeling and Transformation System [7] is modeling environment that allows creating and transforming UML models with OCL constraints. It has a visual language to define complex transformations similar to UML activity diagrams. Matching mechanism uses metamodel approach i.e. is looking for a part of a model that can be identified as an instance of a metamodel pattern. Transformation constraints can be defined in OCL and are finally transformed to C# code. The whole system is also implemented in C#, which limits the platforms it is available for. There are no means provided for simplification of the analysis and according to [15] the performance of VMTS turned to be a number of orders of magnitude worse than that of Fujaba.

FUJABA (From UML to Java And Back Again) [8] was initially developed as UML modeling tool with code generation to Java. Later a visual language for model transformation based on graph rewriting was added. Transformations are defined by means of Story Diagrams, which can be seen as a mixture of UML activity and collaboration diagrams. For each transformation Java code implementing it is generated. The framework can be extended by means of plug-ins and the Fujaba itself can be integrated with Eclipse modeling environment [12]. Support of model constraint analysis is also absent.

The only tool that provides some model analyzing capabilities, which might be helpful when implementing optimizing transformations, is MagicDraw with ParaMagic plug-in [11]. However it uses System Modeling Language (SysML) [10], not UML. Since SysML models can contain parametric diagrams specifying relations e.g. equations among system variables. Mentioned plug-in allows in some cases resolving the equations, but cannot currently work with OCL constraints in the model.

Any of the tools mentioned does not provide all the features needed for convenient implementation of transformations optimizing model performance. It means that this issue should be studied further in detail before the final decision is made, but a new solution designed with support of optimizing transformation in mind is likely to be created by authors in case no other tool is found to meet all the requirements.

## III. Causes for Model Defects

There can be many reasons why UML models may be optimized. The most common case is a mistake of a user creating the model. Modeling languages have higher level of abstraction compared to those of traditional programming languages and therefore operate with the concepts that are closer to the problem domain, not to the programming language domain. Even users that do not have a professional knowledge in programming, but have it in the problem domain can develop software systems with MDD. However such users are more likely to make mistakes in design and implementation and hence should be supplied with the tools detecting and preventing them.

Another typical situation emerges in case of component reuse. The reuse of components from other systems or component libraries can save a lot of time and effort, but at the same time can lead to ineffective or redundant models. This drawback may be overcome if there is an optimizer that can transform the components being reused in the system into more effective ones taking into account the semantics of the system being created. For example if we consider a component that implements the process of organization of computers into a tree-structure according to standardized protocol. Tree structures are used for example to implement multicast functionality in computer networks. Protocols standards usually describe several roles that participants can play during the interactions according to the protocol. For a tree-structure organization there are three basic roles a computer modeled as a class with a state machine can play in the interaction: newcomer – a computer that would like to join a multicast tree; root – a computer that accepts join requests of the newcomers and ex-son – a computer that was a part of the tree, but needs to find a new parent because of tree reorganization [16]. A reusable component implementing this protocol should cover all the roles and cases described in the standard. However when being used in a particular system such implementation can be redundant if for example a model contains constraints that limit protocol implementation to certain roles only. For instance a certain system can make ex-son roles to be impossible. An optimizer in this case should be able analyze the constraints, detect, and remove statemachine elements needed only for implementation of redundant roles.

Behavioral features of UML models can even be generated automatically, for example on the basis of the formal specification or a complete set of test cases. Generated models also need to be checked for their performance, since often there is a lot of space for improvement.

## IV. Tool Support

The transformations we study can be implemented as an extension to existing integrated modeling environment like for example Eclipse Modeling Framework [12]. The module can be divided into two parts Analyzer and Transformer as shown in fig. 1. The following workflow looks natural when working with the extension. When one wants to optimize the model he activates the corresponding command in the modeling environment. This can be done automatically when code generation is performed. Analyzer then checks the model and reasons about contained constraints.

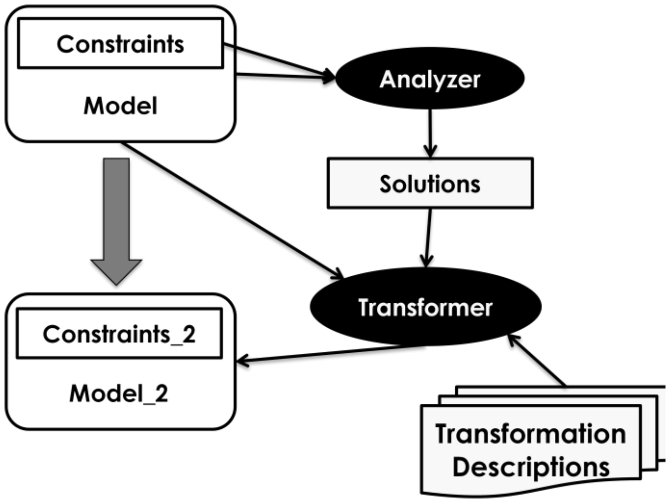


Fig.1. The Scheme of Optimizers Work

The purpose of analyzer's work is to provide additional information that might be helpful for optimization. The principles of its work are similar to those of partial evaluators [13]. In the beginning as a feasibility study we limit the types of supported constraints to algebraic expressions using operation parameters and class attributes e.g. *self.salary > 0*. Despite visible simplicity according to [14] such constraints are quite common in real systems. Analyzer iteratively propagates constraints over UML model actions. For example, if an input parameter *x : Integer* of an operation is constrained to be in range [0;c] and the first action of the operation declares a local variable *y*, which is initialized as *2x*, then the constraint can be propagated to that statement and limit the values of *y* to be within the interval [0;2c]. As a result of analyzer's work all actions in the model get a set of associated constraints. These results are available to Transformer module.

Transformer contains a set of transformation descriptions. For user convenience this set should be as flexible as possible, i.e. a user should be able to include and exclude transformations from that set. Moreover it is highly desirable that a user can create new transformations from scratch or by combining already existing transformations. A description of the transformation contains a pattern that is matched against user model and constraints defined on this pattern that must be satisfied. The patterns are defined on a metalevel that makes them independent on the model they are matched with. Therefore the matching process is not a search for a part of the model that is isomorphic as multigraph to the pattern being matched, but a search for a part of the model that is an instantiation of the metamodel pattern. In case all the pattern constraints are observed the transformation is added to the list of possible operations. After the matching for all active transformations is completed a user is presented with the list of possible operations for review and confirmation. In order to avoid undesired changes, e.g. those caused by a mistake in constraints, a user should be able to easily find out which constraints in the model made certain transformation possible. It is also important to keep the history of transformations for convenient use; this will allow reverting changes later if requested by user.

For a feasibility study the transformation that removes dead branches from the condition action can be considered. The pattern of this transformation matches all the choice pseudo states of state machines in the model. The constraint of this transformation should state that the estimated by Analyzer range for the expression on which the decision is based intersects with the only decision answer range. In this case all other answer transitions can be removed from the model as they are never fired. Transformation for decision nodes from activities specifications is defined similarly. The ways of formal specification of such transformations are currently under investigation by the authors.

## V. Conclusion and Future Work

The spread and adoption of MDD by the industry of software development not only requires availability of the tools supporting MDD, but effective execution of the models being created with such tools, therefore transformations that can optimize performance of UML models are highly demandable.

Current results of our research include the preliminary analysis of available tools supporting UML model transformation and the ways optimizing transformation can be formally described. The work will be continued in the following directions: new optimizing transformation will be created; model transformation tools study should be completed to decide the best implementation way; and the effectiveness of the transformations application will be studied on real industry projects.

## REFERENCES

[1] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure. http://www.omg.org/docs/formal/09-02-02.pdf

[2] Object Management Group. Object Constraint Language http://www.omg.org/docs/formal/06-05-01.pdf

[3] S. Muchnick. Advanced Compiler Design and Implementation. Morgan Kaufmann, 1997.

[4] M. Fowler. Refactoring. Improving the Design of Existing Code. Addison-Wesley, 1999.

[5] D. Balasubramanian, A. Narayanan, C. vanBuskirk, G. Karsai. The Graph Rewriting and Transformation Language: GReAT. The proceedings of the Third International Workshop on Graph Based Tools, 2006.

[6] Mens, T., N. Van Eetvelde, S. Demeyer and D. Janssens, Formalizing refactorings with graph transformations, Int'l Journal on Software Tools for Technology Transfer 17 (2005), pp. 247–276.

[7] Lengyel, L., Levendovszky, T., Mezei, G., Charaf, H.: Model transformation with a visual control flow language. International Journal of Computer Science 1(1) (2006) 45–53.

[8] FUJABA Homepage, http://wwwcs.upb.de/cs/fujaba/

[9] B.W. Watson, A taxonomy of finite automata minimization algorithms, Eindhoven University of Technology, The Netherlands. Computing Science Note 93/44 (1993).

[10] Object Management Group. OMG Systems Modeling Language. http://www.omg.org/docs/formal/08-11-02.pdf

[11] InterCAX SysML Parametric Solvers. http://www.intercax.com/sysml

[12] Eclipse Modeling Framework Project (EMF) http://www.eclipse.org/modeling/emf/

[13] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. Partial Evaluation and Program Generation. PrenticeHall, 1993.

[14] Wahler, M., Ackerman, L., Schneider, S.: Using IBM Constraint Patterns and Consistency. Analysis. IBM Developer Works, May 2008.

[15] A. Rensink, A. Dotor, C. Ermel, S. Jurack, O. Kniemeyer, J. de Lara, S. Maier, T. Staijen, and A. Zündorf. Ludo: A Case Study for Graph Transformation Tools. In A. Schürr, M. Nagl, and A. Zündorf, editors, Applications of Graph Transformation with Industrial Relevance, Proceedings of the Third International AGTIVE 2007 Symposium, volume 5088 of LNCS, Heidelberg, 2008.

[16] Dolejs, O. Tree Building Control Protocol - State Machine. Prague: CTU, Faculty of Electrical Engineering, Department of Control Engineering, 2001. K335/01/210. 9 s.

# Creation of Automaton Classes from Graphical Models and Automatic Solution for Inverse Problem

Yuri Gubin
*DataArt*
*ygubin@dataart.com*

Kirill Timofeev
*DataArt*
*ktimofeev@dataart.com*

Anatoly Shalyto
*SPbSU ITMO*

## Abstract

*Graphical models – integral part of any program development. Using graphical models facilitates analysis of their architecture and understanding of logic.*

*This work shows how to automatically transfer automaton code into graphical isomorphic model, using Java programming and DOT modeling languages.*

## 1. Introduction

Currently in the design of programs with complex behavior actively used tools for graphical representation of logic and structure, for example, UML[1]. They make it possible to analyze the program, working with its components, rather than source code. The graphical presentation makes it easy to debug, and provides the means to address possible shortcomings and mistakes.

Automaton library for graphical representation of state machine will be developed in this work using language with static types Java. It should to provide the possibility of creating automaton classes and automatically transfer them to isomorphic graphic model. This library implements the domain-specific language (DSL, Domain Specific Language [2]), which will allow:

- to check and modify the program without any knowledge of Java programming language by experts of the subject area;
- to provide failure–resistance of developed program;
- to work with the code using terms of subject area - to improve the readability of source code.

The library has been developing for object-oriented programs with explicit allocation of states and state machine to describe behavior of them [3].

The purpose of this work – the implementation of the automatic transfer of automaton classes (automated classes [3]) executable code to isomorphic graphical model (reverse engineering). As an example, automaton of user registration on Web site from the Restful-authentication plug-in [4] will be implemented.

## 2. Description of the algorithm of automatic transfer of the executable code to isomorphic graphical model

The proposed algorithm converting the following components of a state machine to a graphical representation:

1) states that do not belongs to any group;
2) groups, and nested groups;
3) nested states and transitions.

Components of state machine can be read using meta information of automaton class, or easily from arrays of elements.

The result of the algorithm will be a model in text language DOT [5]. This language provides ability to describe different graphical components (applied to automaton – states, transitions and groups). Also it allows editing styles of components.

As an example, a model of two parallel processes described in this language:

```
digraph G {// Name of oriented graph
 subgraph cluster0 {// process #1
  // Styles
  node [style=filled,color=white];
  style=filled;
  color=lightgrey;
  // Nodes and
  a0 -> a1 -> a2 -> a3; arcs
  // Name of the group
  label = "process #1";
 }
 subgraph cluster1 {// process #2
  node [style=filled];  // Style
  // Nodes and arcs
  b0 -> b1 -> b2 -> b3;
  // Name of the group
  label = "process #2";
  color=blue   // Border color
 }
 // Nodes and arcs
 start -> a0;
 start -> b0;
 a1 -> b3;
 b2 -> a3;
 a3 -> a0;
 a3 -> end;
 b3 -> end;
 // Styles for the initial and final
 // nodes
start [shape=Mdiamond];
end [shape=Msquare];}
```

Dot (dot) utility from graphviz [6] package is using to get the image from the DOT model description.

Figure 1 shows a graphical representation of the model obtained from the description in DOT language shown above.

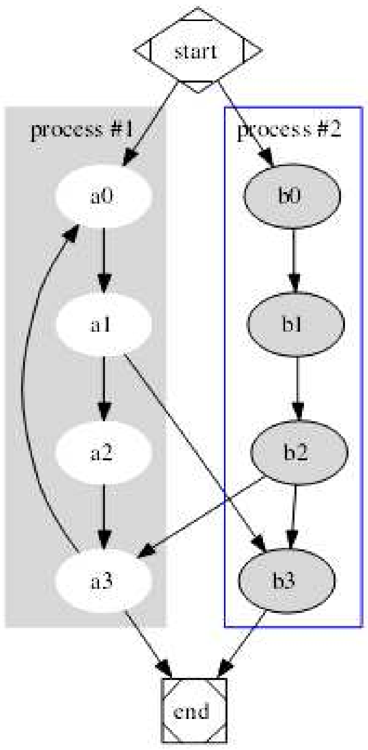


**Figure 1. An example image described using the DOT language**

It is known how to transform each component of automaton class into DOT language. States, groups and transitions can be represented as a formatted string. So, we need to process states firstly, then groups and nested states and later transitions. Using this order we would not create duplicated nodes in DOT model and result will be easy to use.

For example transition could be transformed into `start -> a0`; where `start` and `a0` is a names of states. State could be saved just as `a0` and group could be saved as

```
subgraph clusterGroup{
label=”name of group”
;}
```

Getting presentation in DOT language from all components we can create an output model.

## 3. The implementation of the library to create an object-automaton programs

To ensure compliance with the objectives develop the automaton library, which will allow effectively implementing of the automaton classes firstly, and, secondly, will contain a class that provides automatic transfer of executable code to the graphical isomorphic model. The development of automaton library is taken into feature of further conversion of automaton Java classes into DOT text description.

The developed library includes the following classes:

- State – class, provides ability to specify functions at the entrance to the state and at the exit from it;
- Transition – transition between states and groups. Allows to specify functions during transition;
- StateGroup – group of states.
- DSL – base class for automaton classes. This class contains all needed common functionality for automaton class (i.e. methods for event reading, default constructors and etc.).The DSL class also includes method Compile for setting up initial state of automaton using metadata of automaton class.

Classes State, Transition and StateGroup inherited from common class Entity. This class provides basic properties, which allow uniformly processing of all components.

This library published in «Projects» section on http://is.ifmo.ru.

Creation of automaton classes for application is based on the library classes. Each of such automaton classes includes described method and all needed common functionality.

Feature of the developed library is that it uses anonymous classes for implementing functions in states and transitions. Anonymous class – is a local class without name. It has been creating and initializing in single expression [7]. This allow us to use them instead of lambda functions of Ruby language [4] for creating states and transitions with all needed functions without separated declaration of methods. Interfaces Guard and Action have been using to create described anonymous classes.

As an example, consider the graphical model of an automaton (Figure 2).

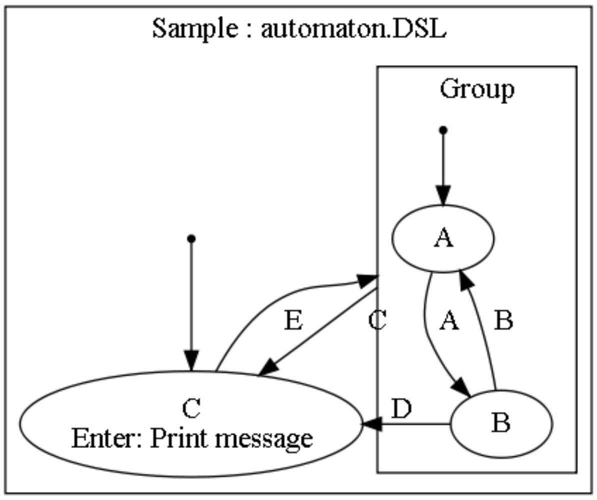


**Figure. 2. Model of an automaton.**

Create Java automaton class for proposed model with developed library.

```
import java.util.*;
import automaton.*;
public class Sample extends DSL{
  public State c = new State("C",
// Anonymous class for event on state
// entrance
     new Action() {
       public void go(){
         System.out.println(
           "Come to state C");}
       public String toDOT() {
         return "Print message";}
     },
     null,null); // No other events
  public StateGroup group =
    new StateGroup("Group");
  public Transition tgc =
    new Transition("GC","C",group,c);
  public Transition tcg =
    new Transition("CG","E",c,group);
// Will be created in constructor
  public Transition tbc;

// Constructor
  public Sample () {
    Vector <Entity> groupEntity =
      new Vector<Entity>();
    State a = new State("A");
    State b = new State("B");
    groupEntity.add(a);
    groupEntity.add(b);
    groupEntity.add(
      new Transition("AB","A",a,b));
    groupEntity.add(
      new Transition("BA","B",b,a));
     tbc = new
Transition("BC","D",b,c);
     group.setAll(groupEntity);
  }
}
```

Note that separated method for state entrance is not required in this code, because code was developed using anonymous class with implemented needed logic. Using of developed library provides:

- creation of methods for states and transitions by means of anonymous classes without duplicated separated methods [8];
- inheritance of automaton classes without additional tools;
- syntactic attractiveness.

## 4. Implementation of automatic isomorphic transfer algorithm in the library

In previous section it was shown how to create an automaton class with developed library. Proceed to consider the inverse problem.

Consider this algorithm on an example of automaton of user registration on Web site from the Restful-authentication plug-in. User registration consists from follow actions:

- fill the registration form;
- enter an activation code;
- fill the personal information.

Administrator can perform follow action:

- delete user and it personal information from system.

Figure 3 shows the expanded graphical notation [9], which describe automaton for registration.

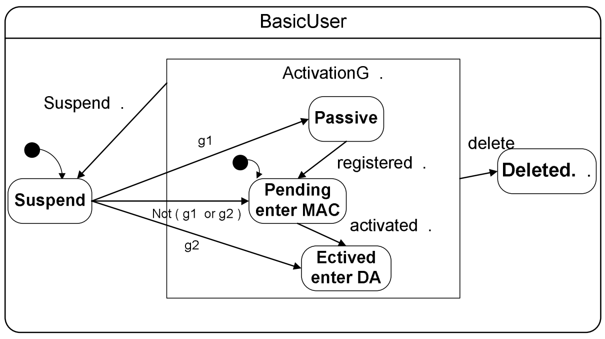


**Figure 3. Automaton for registration**

This automaton includes the following states:

- Suspended – user is waiting;
- Pending – user should to confirm registration by activation code;
- Active – user registered;
- Passive – user only logged in registration system;
- Deleted – user removed.

Group «ActivationG» contains states, in which user acts with registration system.

For its creation, it is necessary to define the collection of states and transitions for group. Determination of the group's structure occurs when the defined collection is passed to the constructor of group. Creation of group placed in constructor of automaton class, so group keeps it structure during inheritance.

Create automaton class BasicUser (described on Fig. 3):

```
import java.util.*;
import automaton.*;
public class BasicUser extends DSL{
  public State deleted =
    new State ("Deleted");
  public State suspended =
    new State ("Suspended");
  public StateGroup activation =
    new StateGroup("ActivationG");
  Vector <Entity> activationStates =
    new Vector <Entity> ();
  public Transition
          activation_suspend =
    new Transition(
     "Activation_suspend", "suspend",
     activation, suspended);
  public Transition
          activation_delete =
    new Transition(
      "Activation_delete", "delete",
      activation, deleted);
  public Transition suspend_passive;
  public Transition suspend_pending;
  public Transition suspend_active;
  void createActive() {
   State stateA = new State(
      "Pending enter mac");
   activationStates.add(stateA);
   State stateB =
       new State ("passive");
   activationStates.add(stateB);
   State stateC = new State (
       "Active enter DA");
   activationStates.add(stateC);
   Transition tran1 = new Transition(
     "passive_pending", "register",
      new Guard() {
        public boolean is() {
          return true;}
      public String toDOT() {
          return "guard";}},
      null, stateB, stateA);
   activationStates.add(tran1);
   Transition tran2 = new Transition(
     "pending_active", "activate",
     new Guard() {
       public boolean is() {
         return true;}
       public String toDOT() {
         return "";}},
       null, stateA, stateC);
   activationStates.add(tran2);
   suspend_passive = new Transition(
     "suspended_passive",
     "unsuspended",
     new Guard() {
     public boolean is() {
       return true;}
     public String toDOT() {
    return "not (guard1 or guard2)";}
  },
  null, suspended, stateA);
  suspend_pending = new Transition(
    "suspended_pending",
    "unsuspended",
    new Guard() {
      public boolean is() {
        return true;}
      public String toDOT() {
        return "guard1"; }
    },
    null, suspended, stateB);
suspend_active = new Transition(
  "suspended_active",
  "unsuspended",
   new Guard() {
    public boolean is() {
      return true;}
    public String toDOT() {
      return "guard2";}
  },
  null, suspended, stateC);
  activation.setAll(activationStates);
 }
   public BasicUser () {
     super () ;
     createActive () ;
   }
 }
```

Implementation of algorithm for transform to isomorphic graphic model requires additional methods for all classes, used in automaton class. As described in section 2, each component class should contain method for transforming it to DOT. More over this each actions and guard conditions should to be converted to DOT too.

Implementation of the algorithm:

```
public String toString () {
  String res = "";
  String subgraph = "";
// Setup header
  res = "digraph " + "veryuniqname" +
" {\ncompound=true;\n";
// state, groups - extracted using
// reflection from automaton class
// method toString of State,
// Transition and StateGroup convert
// them to DOT representation
  for (State state:states) {
    subgraph+=state.toString(mode); }
  for (StateGroup state:groups) {
    subgraph+=state.toString(mode); }
// get all transitions described in
// class
  for (StateGroup sg:groups) {
transitions.addAll(sg.subtransition);
    getAllSubTransitions(sg);
   }
  for (Transition t:transitions) {
    subgraph += t.toString()+"\n";
  }
// adding border for model
  subgraph  = "subgraph cluster0{\n"
+ subgraph + "\nlabel=\" "+dsl_name+"
\"}";
   res+=subgraph;
   return res;
  }
```

Get DOT text description of automaton class by developed method DSL.saveToFile, which implements algorithm of automatic transfer of code. Figure 4 shows image generated by dot utility from gotten text description.

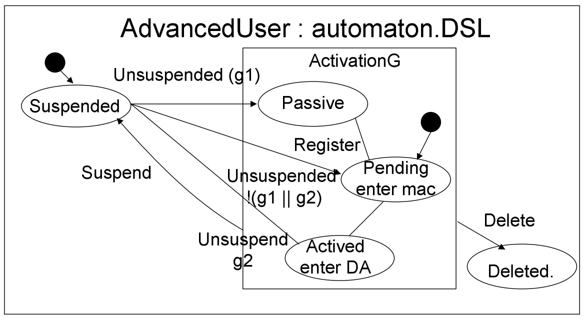


**Figure. 4. Graphical model of BasicUser automaton.**

## 5. Analysis of image data obtained by the algorithm of automatically isomorphic transfer

Practical interest is in the reverse engineering of complex inherited automaton classes.

Consider an automatic isomorphic transfer of automaton class AdvancedUser inherited from BasicUser automaton class. Automaton class AdvancedUser is bringing to registration follow functionality:

- to avoid spam registrations user should to type value of CAPTCHA;
- administrator can block user;

To do this, add the following groups:

- Activation – includes group Active, extended by «Captcha» state;
- Deleted – includes two states: «Blocked» – for blocked users and «Deleted» – for deleted users.

Graphical notation for described automaton shown on figure 5:

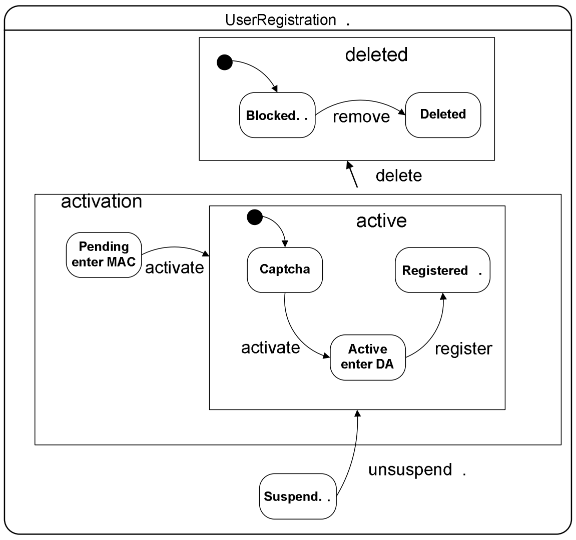


**Figure 5. Extended automaton for registration.**

Application of the developed algorithm and a method for automatically isomorphic converting for implemented AdvancedUser automaton class provides graphic image presented in figure 6.

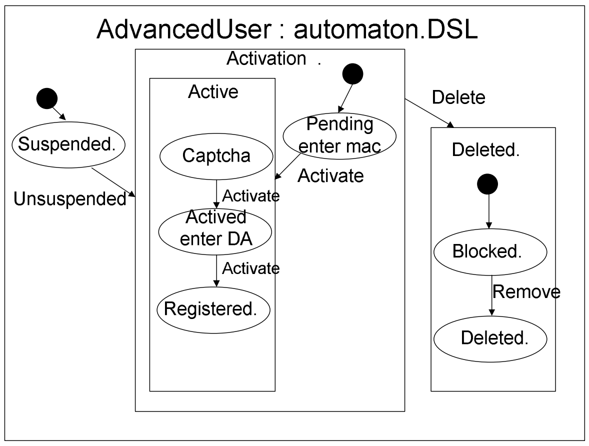


**Figure 6. Graphical model of AdvancedUser.**

Model obtained by automatic isomorphic transfer (Fig. 6) in line with earlier developed model (Fig. 5)

Thus, the solution of transfer code to graphical isomorphic model problem provides an additional analytical tool for debugging of existing logic, introducing additional functionality or inheritance of automaton classes.

## 6. Conclusion

Library for automaton classes creation had been developed in this work. The library allows:

- inherit automaton classes and nest groups;
- create methods for states and transitions (also guard conditions);
- eliminate duplication of code and multiply separated methods by using anonymous classes.

Also algorithm of automatic transfer of automaton classes executable code to isomorphic graphical model had been developed. Automaton of user registration from Restful-authentication plug-in developed and described transfer of them into graphical model (Fig. 6). It was shown that the initial model of automaton corresponding to a graphic representation, obtained by library's function for developed automaton class.

The results of this work will be used in further studies:

1) dynamic and static verification of automata displaying counterexamples in visual form, using the reverse engineering;

2) the establishment of libraries, which will facilitate the testing process of automaton classes.

## 7. References

[1] F. A. Novikov *Visual design of programs*, «Information and control systems». 2005. # 6, http://is.ifmo.ru/works/visualcons/

[2] T Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Texas: Pragmatic Bookshelf, 2007.

[3] N. I. Polikarpova, A. A. Shalyto *Automaton programming*. SPb.: Piter, 2009. http://is.ifmo.ru/books/_book.pdf

[4] A. A. Astafurov, K. I. Timofeev, A. A. Shalyto, *Automata Classes Inheritance in Dynamic Language Ruby* , Software Engineering Conference (Russia) 2008**.** http://www.secr.ru/etc/secr2008_artyom_astafurov_automata _classes_inheritance_in_ruby.pdf

[5] E. Gansner, E. Koutsofios, S. North.*Drawing graphs with dot*. http://www.graphviz.org

[6] Graphviz package. http://www.graphviz.org

[7] *Academy of Modern Programming.* http://www.amse.ru/courses/oopjava/12.php

[8] E. A Zayakin., A. A. Shalyto, *The method of eliminating repetitive code snippets in the implementation of finite automata*. SPbSU ITMO, 2003. http://is.ifmo.ru/projects/life_app/

[9] D. G. Shopyrin, A. A. Shalyto *The graphical notation of inheritance automaton classes.* «Programming»" 2007. # 5 http://is.ifmo.ru/works/_12_12_2007_shopyrin.pdf

# Modeling Security Threats to Cryptographically Protected Data

Alexandra A. Savelieva

Scientific Advisor: Prof. Sergey M. Avdoshin

**In this paper, we introduce a mathematical model of threats for analyzing the security of cryptographic systems based on risk management principles. We also provide economic indicators as a basis to build a rationale for investments to cryptographic systems. Some points of designing software tools to support our methodology are covered in the paper. The new approach that incorporates the threat model, automatic cryptographic strength verification tools and economic techniques, is instrumental for providing sound arguments to choose a cryptographic system and for implementing an information security strategy. An overview of alternative approaches is provided along with the results of comparative analysis revealing their drawbacks as compared to the method presented in this paper.**

***Index Terms*—risk management, threat modeling, cryptographic system, discounted cash flow.**

## I. Introduction

Ross Anderson, Professor in Security Engineering at the University of Cambridge Computer Laboratory and an industry consultant, concludes his well-known paper [2] saying *"the evaluator should not restrict herself to technical tools like cryptanalysis and information flow, but also apply economic tools"*. Our paper aims at providing a formal way of analyzing cryptographic systems security.

The analysis of modern publications on security revealed a lack of methods designed to facilitate the process of cryptographic protection efficiency. Formalized security risk analysis and management methodologies such as CRAMM [10], RiskWatch [24] and GRIF [11] are focused on information system security as a whole and do not consider the peculiarities of evaluating cryptographic systems. There is a mathematical model designed by V.P. Ivanov [16] which applies the principles of the catastrophe theory and queuing theory to computing of a cryptographic system efficiency indicators. Although the approach incorporates economic and technical perspective, its major restriction is that it can only apply to the so-called *restricted-use cryptographic systems* [7] whose security depends on keeping both the encryption and decryption algorithms secret. The author reduces the problem of breaking a cipher to engineering analysis of the program that implements the encryption mechanism. This assumption is inadmissible for modern cryptographic systems, being in conflict with Kerckhoffs's fundamental principle [17] that encryption should not depend on the secrecy of the system - which sooner or later would be compromised - but should solely depend on the secrecy of the key. Finally, various tools for cryptographic protocols analysis [5, 6, 8] focus only on the high-level, conceptual design of a protocol on the supposition that cryptographic algorithms satisfy perfect encryption assumptions, so the strength of ciphers remains out of scope.

## II. Problem Statement

The purpose of our work is to develop an approach to analyzing the security of cryptographic systems. In order to achieve the goal, we need to:

1) formulate the steps of cryptographic systems evaluation process;
2) develop a mathematical model of security threats;
3) design software tools to facilitate the process of cryptosystem efficiency assessment by a computer security specialist;
4) select appropriate economic indicators as a basis to build an economic rationale for investments to cryptographic systems and to provide sound arguments for implementing an information security strategy.

Results of the 1st stage of our research were published in proceedings of SYRCoSE'2008 [26]. In particular, items (1) and (3) from the above list were considered. In addition to it, in [26] we described our multiple-category divisions of cryptographic systems, adversaries and attacks designed for developing of a mathematical model of security threats (2). Therefore, in this paper we will focus on new results on (2) and (4) achieved ever since; as for item (1), we will restrict ourselves to providing a brief overview. We have also decided it is appropriate to elaborate more on (3) since some important aspects of designing new tools for cryptanalysis did not receive much attention in the previous paper.

## III. Cryptographic Systems Evaluation Process

The process of cryptosystem efficiency assessment can be described as a sequence of steps, each of them directed at answering a specific question [26]:

- Step 1: What cryptosystem is the object of attack?
- Step 2: Who wants to attack the cryptosystem?
- Step 3: Which attack techniques are most likely to be used to break the cryptosystem?
- Step 4: Is the cryptosystem capable of withstanding such attacks?
- Step 5: Does the cryptosystem provide sufficient security in the given context?

The environment typically imposes restrictions on the attack scenarios that the cryptographic systems are exposed to, so Steps 1 to 3 imply modeling threats to a cryptographic system in a given context. Step 4 is about analyzing the cryptographic system resistance to the types of attacks defined at Step 3. Finally, Step 5 involves using various risk analysis techniques and economic tools to evaluate the data obtained during Steps 1-4.

## IV. ABC-Model of Security Threats

We can assume that the adversary is most likely to choose the attack with the maximum benefit for a given cost, or choose the least costly attack that gives them a particular benefit [27]. Each cryptosystem has a set of attacks that is applicable to it and a set of attacks that is not. These hypotheses perfectly fit into common risk-management methodologies and result in the following approach to evaluating security threats.

Each cryptanalytic attack has a value of risk associated with it and defined as the product of probability of the hazard and its potential impact:

*Risk = Probability · Impact*

*Impact* refers to effect of an attack on a specific type of cryptographic system. *Probability* reflects the likelihood that an adversary will consider a specific type of attack appropriate in terms of available resources and target secret data. Thus, a formal model of the cryptosystem coupled with formal models of the adversaries will yield a set of the most hazardous attacks that the cryptosystem is exposed to. The model of security threats represented as a composition of 3 elements will be referred to as an *ABC-model* ('A' for **a**ttack, 'B' for code**b**reaker and 'C' for **c**ryptosystem). In [26], a description of multiple-category divisions of cryptographic systems, adversaries and attacks that we suggest as a basis for modeling the components of a security threats is provided.

Let $\mathrm{A} \subseteq A_1 \times A_2 \times ... \times A_9$ be a set of parametric models of attack, where $A_i$ ($i = \overline{1,9}$) represents a domain for the ith parameter as per our taxonomy [26]. Each model $\vec{a} \in \mathrm{A}$ is a vector $\left(a_1, a_2, \dots, a_9\right)$, where $a_i \in A_i$.

Similarly, a parametric model for a code-breaker is $\vec{b} \in \mathrm{B}$, where $\mathrm{B} \subseteq B_1 \times B_2 \times ... \times B_6$, $B_j$ ($j = \overline{1,6}$) represents a domain for the jth parameter, and parametric model for a cryptographic system is $\vec{c} \in \mathrm{C}$, where $\mathrm{C} \subseteq C_1 \times C_2 \times ... \times C_6$, $C_k$ ($k = \overline{1,6}$) represents a domain for the kth parameter as per our taxonomies. It is important to note that sets $A_i$, $B_j$, and $C_k$ are finite.

For simplicity, we will further omit the word 'model' when referring to parametric models of attacks, codebreakers and cryptosystems.

Let $\Re : \mathrm{A} \times \mathrm{B} \times \mathrm{C} \to [0; 1]$ be a function defining the level of risk associated with an attack $\vec{a} \in \mathrm{A}$ as applied by a code-breaker $\vec{b} \in \mathrm{B}$ for cryptanalysis of a cryptosystem $\vec{c} \in \mathrm{C}$. Let function $\mathrm{I} : \mathrm{C} \times \mathrm{A} \to [0; 1]$ define impact (as described above), and function $\mathrm{P} : \mathrm{B} \times \mathrm{A} \to [0; 1]$ define probability. Then risk $\Re$ is evaluated as follows:

$$\Re(\vec{a},\vec{b},\vec{c}) = \mathrm{I}(\vec{c},\vec{a}) \cdot \mathrm{P}(\vec{b},\vec{a})$$

The function $\mathrm{I}(\vec{c},\vec{a})$ is recursively defined via a family of functions $\mathrm{I}_{gh} : C_g \times A_h \to \mathbb{R}_+$, $g = \overline{1,6}$, $h = \overline{1,9}$, where $\mathbb{R}_+$ is a set of nonnegative real numbers. $\mathrm{I}_{gh}$ defines the level of interference between parameters $c_g$ and $a_h$:

- $\mathrm{I}_{gh}(c,a) = 0$, if an attack with parameter value $a \in A_h$ is inapplicable to a cryptosystem with parameter value $c \in C_g$;
- $0 < \mathrm{I}_{gh}(c,a) < 1$, if a cryptosystem parameter value $c \in C_g$ reduces the likelihood that an attack with parameter value $a \in A_h$ can achieve a success;
- $\mathrm{I}_{gh}(c,a) = 1$, in case of no correlation between parameters $c \in C_g$ and $a \in A_h$;
- $\mathrm{I}_{gh}(c,a) > 1$, if a cryptosystem parameter value $c \in C_g$ points out a high probability that an attack with parameter value $a \in A_h$ will be instrumental for cryptanalysis.

To demonstrate the dependency, an illustrative example will be useful. If a cipher is implemented in hardware, it increases the probability that *side-channel attacks* [30] based on information gained from the physical implementation of a cryptosystem (including timing, power consumption, and electromagnetic leaks) will be used to for cryptanalysis. The quantitative level of interference is defined based on expert knowledge.

Let $\overline{\mathrm{I}_{gh}} : C_g \times A_h \to [0; 1]$ be a normalized function:

$$\overline{\mathrm{I}_{gh}}(c,a) = \frac{\mathrm{I}_{gh}(c,a)}{\sum\limits_{\xi \in C_g} \mathrm{I}_{gh}(\xi,a)}$$

Then the level of damage from an attack $\vec{a} \in \mathrm{A}$ to a cryptosystem $\vec{c} \in \mathrm{C}$ is evaluated as follows:

$$\mathrm{I}(\vec{c},\vec{a}) = \min_{h=\overline{1,9}} \prod_{g=\overline{1,5}} \overline{\mathrm{I}_{gh}}(c_g, a_h)$$

If at least one parameter value contradicts the applicability of $\vec{a} \in \mathrm{A}$ to breaking $\vec{c} \in \mathrm{C}$, the function yields 0: this is achieved through using multiplicative criterion.

Accordingly, $\mathrm{P}(\vec{b},\vec{a})$ is expressed in a similar way via parameters of an attack $\left(a_1, a_2, \dots, a_9\right)$ and a code-breaker $\left(b_1, b_2, \dots, b_6\right)$. An example of correlation between parameters is that a brute-force attack (or any other attack which can be parallelized efficiently) is most likely to be used by an adversary who has access to distributed computation resources.

Therefore, the formula defining the level of risk associated with an attack $\vec{a} \in \mathrm{A}$ as applied by a code-breaker $\vec{b} \in \mathrm{B}$ for cryptanalysis of a cryptosystem $\vec{c} \in \mathrm{C}$ is as follows:

$$\Re(\vec{a},\vec{b},\vec{c}) = \min_{h=\overline{1,9}} \prod_{g=\overline{1,6}} \overline{\mathrm{I}_{gh}}(c_g, a_h) \cdot \min_{h=\overline{1,9}} \prod_{t=\overline{1,6}} \overline{\mathrm{P}_{th}}(b_t, a_h)$$

If the level of risk associated with an attack $\vec{a} \in \mathrm{A}$ in a given context (defined in terms of $\vec{c} \in \mathrm{C}$ and $\vec{b} \in \mathrm{B}$) exceeds a threshold $\theta \in [0;1]$, i.e. $\Re(\vec{a},\vec{b},\vec{c}) > \theta$, then the attack will be considered as a threat that the codebreaker imposes on the cryptosystem. The *admissible risk level* $\theta$ is a customizable parameter of the ABC-threat model. When defining $\theta$, the following two criteria are considered:

- the significance of cryptographically protected data;
- the amount of computing and storage recourses available to the specialist.

In the general case:

- a cryptosystem can comprise a number of sub-systems $\vec{c} \in \mathrm{C}'$ $\left(\mathrm{C}' \subseteq \mathrm{C}\right)$, e.g. a symmetric cipher and a key generator, each of them having a different set of applicable attacks;
- a cryptosystem can be a target for several code-breakers $\vec{b} \in \mathrm{B}' \left(\mathrm{B}' \subseteq \mathrm{B}\right)$ who differ in terms of skills, resources etc.

These assumptions yield a set of attacks $\Lambda = \bigcup_{\vec{b} \in B'} \bigcup_{\vec{c} \in C'} \lambda(\vec{b}, \vec{c})$, where $\lambda(\vec{b},\vec{c}) = \left\{ \vec{a} \in \mathrm{A} : \Re(\vec{a},\vec{b},\vec{c}) > \theta \right\}$. Thus, the process of analyzing a cryptosystem's security is reduced to evaluation of its capability to resist the attacks in $\Lambda$ by means of instrumental tools for cryptanalysis discussed in the following section.

When designing the ABC-model, we stemmed from the following admissions:

- the inaccuracy of expressing the interference between a combination of cryptosystem parameters and a combination of attack parameters through interference between individual parameters is negligible;
- the inaccuracy of modeling code-breakers as independent individuals who are not supposed to cooperate is negligible.

The adjustment of the ABC-model to overcome these admissions would require significant complication of the model. The question of the admissions' influence on the model accuracy is subject to further research.

It is important to note that the taxonomy for cryptanalytic attacks is applicable to modeling attacks not only on cryptosystems but also on cryptographic protocols. This is a very important property of the ABC-model: as shown in [28], the interaction between cryptosystems and cryptographic protocols has not been deeply studied and still remains an open area of research.

## V. SOFTWARE TOOLS FOR CRYPTANALYSIS

Our work on designing a new tool for cryptanalysis was inspired by the need to constantly re-evaluate the cryptographic algorithms strength. Such toolkits already exist for some classes of cryptanalytic attacks and enable finding out vulnerabilities, not only on new cryptographic systems being proposed, but also on old schemes which for long have been considered secure. In particular, in the last years researchers have devoted much effort to develop techniques to formally analyze cryptographic protocols [5, 6, 8]. Another important research direction is that of designing tools for high-level side-channel attack simulation developed with the aim of automating analysis techniques to help a cryptanalyst identify possible implementation vulnerabilities with minimal effort on their side. In [22], an approach is presented based on the SystemC 2.0 language [1], the de facto standard in complex digital system simulations illustrated by a case study of analyzing various implementations of AES [23] algorithm.

In our research, we focused on developing tools for analyzing public-key cryptographic algorithms strength. Before designing a new tool for cryptanalysis, we investigated available solutions for solving discrete logarithms and integer factorization problems which are the basis for various modern cryptographic systems, such as RSA [25] and ElGamal signature scheme [12]. Our analysis is supported by an extensive survey of mathematical libraries [15]. The set of evaluation criteria that we used is as follows:

- The tools should provide efficient implementation of big integer arithmetic operations;
- The tools should be compatible with Windows platforms given the large amount of cryptographic products running under Microsoft operating systems;
- The tools should have a base upon which to write implementations of integer factorization algorithms and index-calculus algorithms for discrete logarithm computation, including algorithms for creating factor bases and linear algebra techniques for solving sparse systems of equations;
- The tools should have extensible architecture so that new methods could be easily added to the implementation with the advent of new cryptanalytic techniques;
- The tools should be completely automatic and should carry out their job even when run by users having a limited amount of expertise in the field.

The advantages of programs like Maple [13] or Mathematica [29] are unlimited precision and easy-to-program algorithms. However, they are extremely inefficient for computations in number theory.

Java also has multiprecision capabilities and is highly portable. However, it is very slow in terms of number-theoretical operations. High performance can be achieved through using of low-level programming languages. Although C and C++ built-in numeric data types have limited precision, there are a lot of multiprecision libraries with many of them available as free software (GNU GPL), e.g. LIP, LiDIA, CLN, NTL, PARI, GMP, MpNT etc.

One of the first multiprecision libraries was LIP (Large Integer Package) [21] written by Arjen K. Lenstra and later maintained by Paul Leyland. Despite being highly portable, the ANSI C library is not appropriate for our purpose as it is not efficient. In addition to it, the library provides no base for developing number-theoretic algorithms.

A Class Library for Numbers (CLN) [9] written by Bruno Haibleand and currently maintained by Richard Kreckel is a C++ library that implements elementary arithmetical, logical and transcendental functions. It has a rich set of classes for integers, rational numbers, floating-point numbers, complex numbers, modular integers etc. The drawbacks of this universality are the lack of emphasis on speed, and hence no optimization for the specific tasks of big integer operations.

GMP (GNU Multiple Precision arithmetic library) [14] was developed by Törbjord Granlund and the GNU free software group. Although this C library for arbitrary precision arithmetic is faster than most multiprecision libraries due to its highly optimized ASM implementations for the most common inner loops and for a lot of CPUs, it has the drawbacks of being incompatible with Windows and lack of primitives to support integer factorization and DLP methods.

LiDIA [20] is a C++ library for computational number theory developed at the Technical University of Darmstadt by Thomas Papanikolau. LiDIA includes highly optimized implementations for multiprecision data types and can use different integer packages (like Berkley MP, GMP, CLN, libI, LIP etc.). LiDIA's drawback is that the library is not portable to Windows platform.

NTL (a Library for doing Number Theory) [19] written and maintained mainly by Victor Shoup is a high-performance C++ library. As shown in [15], NTL outperforms other libraries in terms of big integer operations, however it needs to be extended to become instrumental for our purposes as it has no implementation of either integer factorization or DLP algorithms.

Another disadvantage that all the libraries have in common is the high level of programming skills that a cryptanalyst needs to use them.

Since no alternative solution matches all five criteria at once, the rationale for developing new software tools was clear (see Table I). We designed software tools CRYPTO [3, 4] having in mind the efficiency criteria stated above. Multiprecision C++ library DESIGNER that is the core of CRYPTO is an extension of NTL. To provide the user with an easy access to integer factorization and DLP functions, an application ANALYST was implemented in C#. For illustration of the efficiency of CRYPTO, the timing for 55-bit DLP computation on a 3.2 GHz Intel Pentium/1Gb memory PC is 8 hours against 10 minutes that our implementation takes to compute a discrete logarithm in 80-bit field.

TABLE I

A MULTIPLE-CATEGORY COMPARISON OF AVAILABLE SOLUTIONS FOR SOLVING DISCRETE LOGARITHMS AND INTEGER FACTORIZATION PROBLEMS

| **Alternative** / **Evaluation Criteria** | **Maple** | **LIP** | **CLN** | **LiDIA** | **GMP** | **NTL** | **CRYPTO** |
|---|---|---|---|---|---|---|---|
| **High-performance multiprecision operations** | − | − | − | + | + | + | + |
| **Compatibility with Windows platform** | + | + | + | − | − | + | + |
| **Primitives for modern cryptanalytic algorithms implementation** | − | − | − | + | + | − | + |
| **Extensible architecture** | + | + | + | + | + | + | + |
| **Usability** | + | − | − | − | − | − | + |

## VI. ECONOMIC PERSPECTIVE

We suggest that the *discounted cash flow (or DCF) approach* [18] should be used to provide economic rationale for investments to cryptographic systems. In finance, the DCF is a method of valuing a project, company, or asset using the concepts of the time value of money. All future cash flows are estimated and discounted to give their present values. The discount rate used is generally the appropriate cost of capital and may incorporate judgments of the uncertainty (riskiness) of the future cash flows.

The cash flow $R_t$ related to a cryptographic system can be described using the following formula:

$$R_t = -Cost_t + Profit_t \cdot (1 - \mathrm{P}_t) - Loss_t \cdot \mathrm{P}_t ,$$

where $Cost_t$ is the cost of a implementation, deployment and support of the cryptographic system;

$Profit_t$ is the value of information assets being protected;

$Loss_t$ refers to the hazard in case of unauthorized access to the asset by an adversary;

$\mathrm{P}_t$ is the probability of an adversary to break the cryptographic system;

$t$ is the time (e.g. in years) before the future cash flow occurs.

## VII. CONCLUSION

The paper proposes a formalized methodology for analyzing the efficiency of a cryptosystem. Model-based analysis described in this paper is a part of the five-step process designed to focus on the specific aspects of cryptographic systems security. The methodology is supported by software tools designed to evaluate the cryptographic system capability to resist various types of attacks. We expect that economic perspective introduced in this paper will be of value to security specialists for justifying IT budget and communicating their proposals to the co-workers with financial background.

The direction of our future work is the development a built-in expert knowledge base to aid in-house cryptographic systems expertise. This involves evaluating the dependency between the parameters of a cryptosystem model and the applicable attacks on the one hand, and the parameters of an attacker model and the types of attacks that they are likely to use, on the other hand.

## References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE Standard SystemC Language Reference Manual. Std 1666 - 2005, New York, 2006

[2] Anderson R. Why information security is hard - an economic perspective // Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC '01), 10-14 Dec 2001, New Orleans, Louisiana, USA, 2001.

[3] Avdoshin S.M., Savelieva A.A. Tools for asymmetric ciphers analysis: Industrial registration certificate No. 10193 dated 18.03.2008 (in Russian).

[4] Avdoshin S.M., Savelieva A.A. Tools for asymmetric ciphers analysis:. Certificate of official registration in the register of software No. 2008612526 dated 10.04.2008, Federal Service For Intellectual Property, Patents And Trademarks. (in Russian).

[5] Bodei C., Buchholtz M., Degano P., Nielson F., Riis Nielson H. Automatic validation of protocol narration. In Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW 2003), IEEE Computer Society Press, Washington, 2003. Pp. 126 - 140.

[6] Boreale M., De Incola R., Pugliese R. Proof techniques for cryptographic processes. SIAM J. Comput., 31(3), 2002. Pp. 947-986.

[7] Brassard J. Modern Cryptology. Springer-Verlag, Berlin - Heidelberg, 1988. - 107 p.

[8] Cheminod M., Cibrario Bertolotti I., Durante L., Sisto R., Valenzano A. Tools for cryptographic protocols analysis: A technical and experimental comparison // Computer Standards & Interfaces, 2008.

[9] CLN // Available at: http://www.ginac.de/CLN/ 06.02.2007

[10] CRAMM V Official website // Siemens Enterprise Communications Limited 2006. Available at: www.cramm.com

[11] Digital Security: GRIF //Available: http://www.dsec.ru/products/grif/

[12] ElGamal T. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms // IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985. P. 469-472

[13] Garvan F. The Maple Book. Chapman & Hall/CRC, 2001. - 496 p.

[14] GMP // Available at: http://gmplib.org/ 06.02.2007

[15] Hrițcu C., Goriac I., Gordân R. M., Erbiceanu E. MpNT: Designing a Multiprecision Number Theory Library. Faculty of Computer Science, “Alexandru Ioan Cuza” University, Iasi, 2003.

[16] Ivanov V.P. Mathematical evaluation of information protection from unauthorized access // "Specialnaya tekhnika". 2004, N 1. –Pp. 58-64. (in Russian).

[17] Kerckhoffs A. La cryptographie militaire // Journal des sciences militaires, vol. IX. P. 5-38, Jan. 1883, (P. 161-191, Feb. 1883).

[18] Kruschwitz L., Loeffler A. Discounted Cash Flow: A Theory of the Valuation of Firms (The Wiley Finance Series). Wiley, 2005. 178 p.

[19] NTL: Library for doing Number Theory. Available at: http://www.shoup.net/ntl/ 06.02.2008

[20] LiDIA // Available at: http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/ 06.02.2007

[21] LIP // Available at: http://www.win.tue.nl/~klenstra/ 06.02.2007

[22] Menichelli F., Menicocci R., Olivieri M., Trifiletti A. High-Level Side-Channel Attack Modeling and Simulation for Security-Critical Systems on Chips // IEEE Transactions on Dependable and Secure Computing,Volume: 5, Issue: 3, July-Sept. 2008, Pp. 164-176.

[23] RIJNDAEL description. Submission to NIST by Joan Daemen, Vincent Rijmen // Available at http://csrc.nist.gov/encryption/aes/round1/docs.htm

[24] RiskWatch Official website // RiskWatch, Inc. Available at: http://www.riskwatch.com/

[25] Rivest R.L., Shamir A., Adleman L.M. A Method for Obtaining Digital Signatures and Public Key Cryptosystems// Communications of the ACM, v. 21, n. 2, February 1978. P. 120-126.

[26] Savelieva A. Formal methods and tools for evaluating cryptographic systems security // St. Petersburg, ISP RAS, In Proceedings of the Second Spring Young Researchers’ Colloquium on Software Engineering (SYRCoSE’2008), 2008, Vol 1. ISBN 978-5-91474-006-8. Pp. 33-36.

[27] Schneier B. Beyond Fear. Thinking Sensibly about Security in an Uncertain World. Copernicus Books (September 2003)

[28] Verma R. Protocol Specification and Verification. Lectures on COSC 6397 – Information Assurance. University of Houston, 2006. Available at: www2.cs.uh.edu/~rmverma/M2L1.ppt

[29] Wolfram S. The Mathematica Book, 4th edition. Cambridge University Press and Wolfram Media, Cambridge, 1999, 1470 p.

[30] Zhou Y., Feng D. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing // Physical Security Testing Workshop (Hawaii, September 26-29, 2005. Available at: http://eprint.iacr.org/2005/388.pdf

Manuscript received March 15, 2009.

Alexandra A. Savelieva is a first-year postgraduate student of Software Engineering Department, State University – Higher School of Economics (e-mail: alexandra.savelieva@gmail.com; optional phone: +7(962)902-4310).

Research supervised by Prof. Sergey M. Avdoshin, Head of Software Engineering Department, State University – Higher School of Economics, Russia (e-mail: savdoshin@hse.ru; optional phone: +7(495)771-3238).

# Clustering Algorithms Meta Applier (CAMA) Toolbox

Dmitry S. Shalymov
Mathematics & Mechanics department
Saint-Petersburg State University
Email: shalydim@mail.ru

Kirill Skrygan
Mathematics & Mechanics department
Saint-Petersburg State University
Email: kirillskrygan@gmail.com

Dmitry Lyubimov
Mathematics & Mechanics department
Saint-Petersburg State University
Email: ancient.punk@gmail.com

***Abstract*—Clustering is used in many fields, including machine learning, data mining, financial mathematics, etc. There are many different algorithms for cluster analysis. Nevertheless only a few software tools are available to perform these algorithms with user's data sets and to compare results. We propose CAMA software toolbox for the new clustering algorithms research. The main character of this tool is a possibility to load as user's input datasets, as user's algorithms and to compare results with classical methods. Basic principles of CAMA and used technologies are briefly described. Some ideas for new clustering algorithms are also proposed for further development.**

## I. Introduction

Data clustering is the partitioning of a data set into subsets (or clusters), so that the data in each subset share some common trait, often proximity according to some defined distance measure. It is a common technique for statistical data analysis, which is widely used in machine learning, data mining, pattern recognition, image analysis, financial mathematics and bioinformatics.

After thirty years research there were proposed a great amount of approaches. Most of them were developed for a specific problem and are somewhat ad hoc. Those methods that are more generally applicable tend either to be model-based, and hence require strong parametric assumptions, or to be computation-intensive, or both.

Researches face with many difficulties while investigating an efficiency of their new algorithms. Especially it is important when researcher have to prove that his method is the best one for a special kind of data sets.

Unfortunately only a few number of software tools are available to restrict such difficulties. These tools have essential limitations in their functionality. For example it is impossible to load your own algorithms and perform experiments with it. The good example is an open sourced Cluster Validation Toolbox CVAP [1]. ClusterPack [2] is a collection of MATLAB functions for cluster analysis. It consists of the three modules ClusterVisual, ClusterBasics, and ClusterEnsemble. They contain general clustering algorithms as well as special algorithms. But the number of available test data sets is very few. COMPACT [3] is GUI MATLAB tool that enables to compare some clustering methods. Only four clustering algorithms are supported. Data Clustering & Pattern Recognition (DCPR) [4] is MATLAB tool with friendly interfaces. It supports 6 basic clustering algorithms and 4 data sets. The Fuzzy Clustering and Data Analysis Toolbox [5] is a collection of MATLAB functions. It supports 6 clustering algorithms and over 7 clustering validation algorithms. The engine to compare effectivenes of algorithms is also implemented. SOM Toolbox [6] is a software library for MATLAB 5 (version 5.2 at least) implementing Self-Organizing Map (SOM) algorithm. To compare effectiveness of SOM algorithm more than 11 clustering algorithms are supported. The engine for creation artificial data sets is implemented. But there is also no possibility to load and to perform your own clustering algorithms

All these tools are implemented with MATLAB which is the commercial product with quite expensive license. Our toolbox is free software. You can redistribute it and/or modify it under the terms of the GNU General Public License [7].

CAMA toolbox allows to structure the known information about existent clustering algorithms and to research new effective user's methods of cluster analysis.

In Section 2 we introduce basic principles of CAMA and describe main steps of algorithm processing. In Section 3 Kernel module is described, Section 4 is about two modifications of CAMA implementation. Available set of preloaded algorithms and data sets is introduced in Section 5. Main ideas of new clustering algorithm are proposed in Section 6. We conclude in Section 7 by discussing possible extensions of CAMA software toolbox.

## II. Clustering Analysis

The classification of objects according to similarities among them and organizing of data into groups is the objective of cluster analysis. Clustering techniques do not use prior class identifiers. That's why they are among the unsupervised methods. The main potential of clustering is to detect the underlying structure in data, not only for classification and pattern recognition, but also for model reduction and optimization. Different classifications can be related to the algorithmic approach of the clustering techniques: partitioning, hierarchical, graph-theoretic methods and methods based on objective function.

Clustering techniques can be applied to data that is quantitative (numerical), qualitative (categoric), or a mixture of both. In this thesis, the clustering of quantitative data is considered.

Cluster is a group of objects that are more similar to one another than to members of other clusters. In metric spaces,

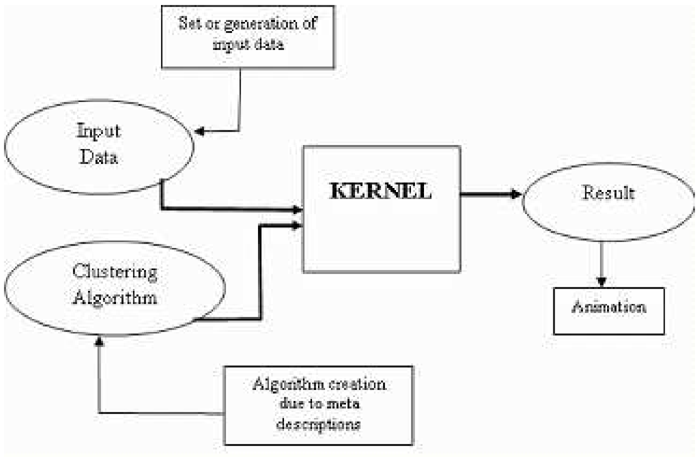

Set or generation of input data
Input Data
KERNEL
Result
Clustering Algorithm
Animation
Algorithm creation due to meta descriptions

Fig. 1. Basic principles of CAMA

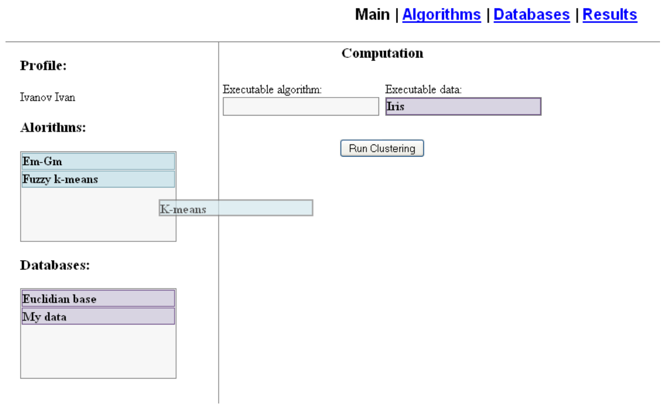


Fig. 2. CAMA Toolbox. Algorithms and data sets drag and drop

similarity is often defined by means of a distance norm. Distance can be measured among the data vectors themselves, or as a distance form a data vector to some prototypical object of the cluster. The prototypes are usually not known beforehand, and are sought by the clustering algorithms simultaneously with the partitioning of the data. The prototypes may be vectors of the same dimension as the data objects, but they can also be defined as linear or nonlinear subspaces or functions.

Since clusters can formally be seen as subsets of the data set, one possible classification of clustering methods can be according to whether the subsets are fuzzy or crisp (hard). Hard clustering methods are based on classical set theory, and require that an object either does or does not belong to a cluster. Fuzzy clustering methods allow objects to belong to several clusters simultaneously, with different degrees of membership.

## III. Clustering Algorithms Meta Applier

The main goal of Clustering Algorithms Meta Applier is the research and approbation of new and existent clustering algorithms on a various number of data sets. CAMA contains an engine to load user's data and user's algorithms, prepared input datasets for experiments and the set of existent algorithms in MATLAB (*.m files) format. CAMA is implemented as a desktop application and as a web-service.

An oracle which knows the correct answer could be used. In that case an accuracy of applied algorithms can be measured due to oracle's knowledge.

Now only hard clustering methods are implemented in CAMA. Other clustering techniques support is a subject for the further development.

Result of algorithm application in CAMA contains the following information: number of clusters in data set, coordinates of cluster centers, number of iterations and character diagrams. It is possible to save extracted results as images and statistic data.

The basic principles of CAMA are shown in Fig.1.

Input data that was generated or was loaded as one of prepared data sets comes to the kernel module with one of the clustering algorithms. Kernel translates algorithm from *.m format and computes the result which is processed with animation module. All GUI interfaces and engines for loading data are implemented with Java. Algorithms processing and calculations are performed with .NET technologies in the kernel which is compiled as DLL.

## IV. Kernel

Kernel is the interpreter of MATLAB language. First *.m files are translated into *.cs files. After that C# compiler compiles it into DLLs. The main problem here is the translation from MATLAB to C# because MATLAB is not the language with pure types. It is partly functional and has a great amount of complex inner functions.

The grammar and semantics are described with ANTLR [8]. The lexical analyzer, parser and additional logics are performed in C# level. Also with C# there were implemented MATLAB functions that are not available in .NET and even in Math.NET Iridium [9].

Interpreter also contains special utility for input data set conversion (usually large number of multidimensional vectors) to the .NET format.

Used technologies: ANTLR and ANLRWorks 1.2.3, MSVS 2008 and .NET 3.5 [10], .NET Reflector [11], Math.NET Iridium.

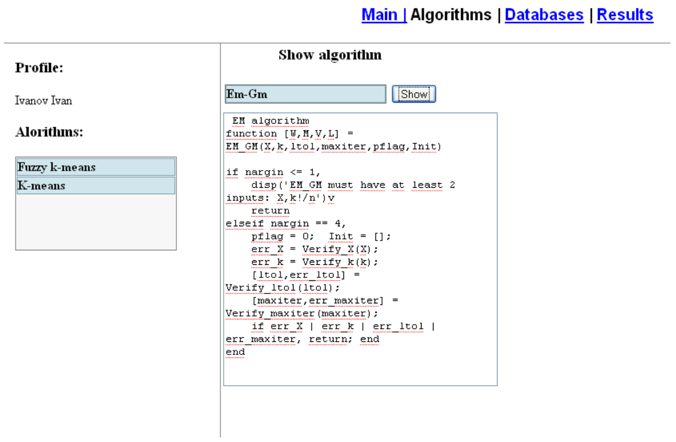
## V. Web-service and desktop application

CAMA is implemented in two modifications: desktop application and web-service application.

All GUI interfaces and modules (except kernel) are implemented with Java technologies.

GUI for desktop and web-service versions are almost the same. Infrastructure of web-service is based on accounts. Each user has its own account for loading and using data sets. After registration user has an access to his personal page which contains personal information, the list of available algorithms and data sets (default data sets and previously loaded by user).

All available algorithms and datasets are visualized so that user can simply drag and drop corresponding items to the evaluation block. It is shown in Fig.2.



Profile:

Ivanov Ivan

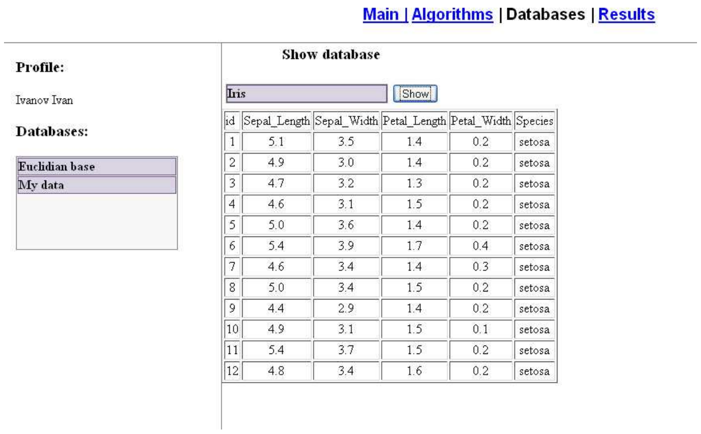
Alorithms:

Fuzzy k-means
K-means

Em-Gm

```
 EM algorithm
function [W,M,V,L] =
EM_GM(X,k,ltol,maxiter,pflag,Init)

if nargin <= 1,
    disp('EM_GM must have at least 2
inputs: X,k!/n')v
    return
elseif nargin == 4,
    pflag = 0;  Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] =
Verify_ltol(ltol);
    [maxiter,err_maxiter] =
Verify_maxiter(maxiter);
    if err_X | err_k | err_ltol |
err_maxiter, return; end
end
```

Fig. 3. CAMA Toolbox. Clustering algorithm text representation



Profile:

Ivanov Ivan

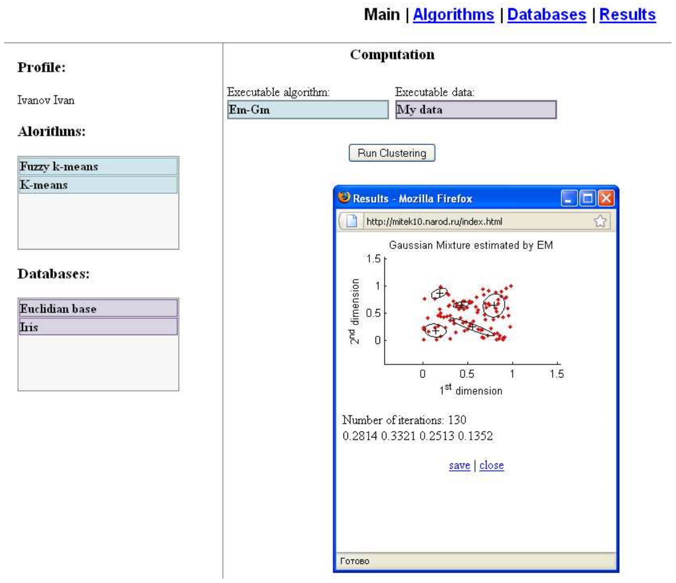
Databases:

Euclidian base
My data

Iris

| id | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width | Species |
|---|---|---|---|---|---|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 6 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 7 | 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 8 | 5.0 | 3.4 | 1.5 | 0.2 | setosa |
| 9 | 4.4 | 2.9 | 1.4 | 0.2 | setosa |
| 10 | 4.9 | 3.1 | 1.5 | 0.1 | setosa |
| 11 | 5.4 | 3.7 | 1.5 | 0.2 | setosa |
| 12 | 4.8 | 3.4 | 1.6 | 0.2 | setosa |

Fig. 4. CAMA Toolbox. Data set representation


Main | Algorithms | Databases | Results
Computation
Run Clustering
Results - Mozilla Firefox
http://mitek10.narod.ru/index.html
Gaussian Mixture estimated by EM
1.5
1
0.5
0
2nd dimension
0
0.5
1
1.5
1st dimension
save | close
Готово

Profile:

Ivanov Ivan

Alorithms:

Fuzzy k-means
K-means

Databases:

Euclidian base
Iris

Executable algorithm: Em-Gm

Executable data: My data

Number of iterations: 130
0.2814 0.3321 0.2513 0.1352

Fig. 5. CAMA Toolbox. Result of clustering artificial data with EM algorithm

User can see the text representation of his algorithms. See Fig.3.

It is possible to load data in csv and text format and check separator between cells in rows. Also user is able to manipulate with data sets by performing SQL queries. Data sets representation is shown in Fig.4.

After execution server will generate a new page with results. The example of clustering artificial data set with EM algorithm [12] can be seen in Fig.5. The new page contains graphics result, number of clusters, coordinates of cluster centers, number of algorithm iterations and other statistics.

Used technologies: Java Server Pages (JSP) [13], Java Servlets (JDK 1.6) [14] and Apache Tomcat Server [15].

In the desktop application multi user's work is not supported. All web-service pages are replaced with dialogs and no one server is implemented.

Used technologies: JDK 1.6 and Java Swing [16].

## VI. Prepared algorithms and data sets

A fundamental, and largely unsolved, problem in cluster analysis is the determination of the "true" number of groups in a data set. Numerous approaches to this problem have been suggested over the years. We decided to start from such kinds of algorithms. This is Calinski and Harabasz's index (1974), Hartigan's rule (1975), the Kranowski and Lai test (1985), silhouette statistic (Kaufman and Rousseeuw, 1990), Gap statistic (Tibshirani, 2001) and "jump" method (Sugar and James, 2003).

There are three important problems in clustering algorithms: determination of the "true" number of clusters, initial values of cluster centers and metrics. Now we implemented algorithms that solve only first problem.

Classical data sets that are used to check consistency of clustering algorithms are available by default. This is Iris flower (Fisher, 1936), Wisconsin (Wolberg and Mangasarian, 1990) and Auto Data (Quinlan, 1993). Many data sets are available in UCI Machine Learning Repository [17]. We suppose to support possibility to load and update data from this portal soon.

## VII. New clustering algorithm

The new clustering method is a modification of Sugar and James algorithm [18] that determines the number of clusters. The main procedure in this algorithm, which is called "jump method", is based on "distortion" that determines a measure of within cluster dispersion. It has the following simple steps:

1. Run the k-means [19] algorithm for different numbers of clusters, $K$, and calculate the corresponding distortions

2. Select a transformation power, $Y > 0$ (A typical value is $Y = p/2$)

3. Calculate the "jumps" in transformed distortion $J_k = d_k^{-Y} - d_{k-1}^{-Y}$

4. Estimate the number of clusters in the data set by $K^* = argmax_k J_k$ the value of $K$ associated with the largest jump.

K-means is a typical clustering algorithm but it has two shortcomings in clustering large data sets: number of clusters

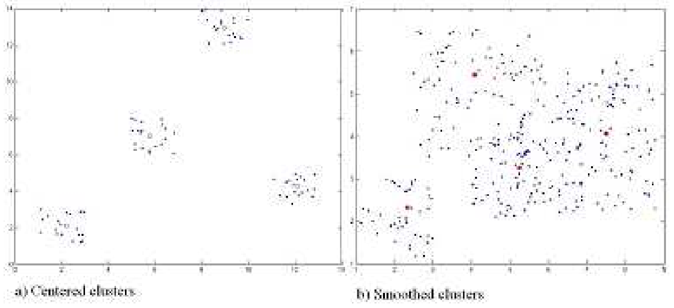


Fig. 6. SPSA algorithm in Sugar-James method. a) centered clusters; b) smoothed clusters

dependency and degeneracy. Number of clusters dependency is that the value of k is very critical to the clustering result. This shortcoming is not essential in our case. Degeneracy means that the clustering may end with some empty clusters. To avoid this problem and to improve results for noisy data it is proposed to use simultaneous perturbation stochastic approximation (SPSA) [20] algorithms which keep appropriate estimations under almost arbitrary noise [21].

The efficiency of modified Sugar-James method is demonstrated in the Fig.6.

Red circles in the Fig.6 show approximated cluster centers. As we could see they are similar with the ”true” values.

There are the other modifications of Sugar-James method concerned with analysis of ”distortion” curve.

This algorithm is easy to implement and to invoke in CAMA toolbox. Now all results were obtained with MATLAB.

## VIII. Conclusion

Described CAMA software tool for clustering data sets with a number of different algorithms is useful for researchers who want to investigate advantages of their new algorithms and for ordinary users who would like to determine the shape of their data sets, extract atypical objects or reduce amount of stored data.

CAMA also could help to gather and to structure known information about existent algorithms and data sets. Having such information it is possible to tune some clustering algorithms for special cases and even investigate the new ones.

For further development we hope to enrich the number of available algorithms not only for determination of the ”true” number of clusters, but also for determination cluster centers with hierarchical algorithms. It is supposed to implement possibility of working with many metrics at once. The engine to load and to update data from UCI [17] portal will be available soon.

Support of multithreading instructions for simultaneous performing algorithms and integration into the GRID portal is also planned.

## References

[1] Cluster Validation Toolbox CVAP, *http://www.mathworks.com/matlabcentral/fileexchange/14620*

[2] ClusterPack MATLAB Toolbox, *http://www.ideal.ece.utexas.edu/ strehl/soft.html*

[3] COMPACT Toolbox, *http://adios.tau.ac.il/compact*

[4] Data Clustering & Pattern Recognition Toolbox, *http://neural.cs.nthu.edu.tw/jang/matlab/toolbox/DCPR*

[5] Fuzzy Clustering and Data Analysis Toolbox, *http://webscripts.softpedia.com/scriptDownload/Clustering-Toolbox-Download-35404.html*

[6] Self-Organizing Map Toolbox, *http://www.cis.hut.fi/projects/somtoolbox/download*

[7] GNU General Public License, *http://www.gnu.org/copyleft/gpl.html*

[8] ANTLR, *http://www.antlr.org*

[9] Math.NET Iridium, *http://mathnet.opensourcedotnet.info/downloads/IridiumCurrentRelease.ashx*

[10] MSVS 2008 and .NET 3.5, *http://www.microsoft.com/events/series/msdnvs2008.aspx*

[11] .NET Reflector, *http://www.red-gate.com/products/reflector*

[12] G. McLachlan and T. Krishnan, *The EM algorithm and extensions*, Wiley, New York, 1997.

[13] Java Server Pages (JSP), *http://java.sun.com/products/jsp*

[14] Java Servlets, *http://java.sun.com/products/servlet*

[15] Apache Tomcat Server, *http://tomcat.apache.org*

[16] Java Swing, *http://java.sun.com/docs/books/tutorial/uiswing*

[17] UC Irvine Machine Learning Repository, *http://archive.ics.uci.edu/ml*

[18] C. Sugar and G. James, *Finding the number of clusters in a data set : An information theoretic approach*, Journal of the American Statistical Association (98), 2003, pp. 750 - 763.

[19] J. A. Hartigan and M. A Wong, *A K-Means Clustering Algorithm*, Applied Statistics(28), 1979, pp. 100 - 108.

[20] J. C. Spall, *Introduction to Stochastic Search and Optimization. Estimation, Simulation and Control*, Wiley, London, 2003.

[21] O. N. Granichin, B. T. Polyak, *Randomized Algorithms of optimization and Estimation under Almost Arbitrary Noise*, Nauka, Moscow, 2003.

# Dynamic web-components and web environment behavior analysis

Suvorov V.[1]

[1]Saint- Petersburg State University, Math and Mechanics faculty

**A problem of automated interaction with complicated website at a user level is considered. The general case of playing online game is considered as a model of any site usage. Three functional components – the user actions, the site logic and other parties' actions are distinguished. A method of reconstructing site components, their behavior and interconnectivity is suggested. Practical results of using the method are shown - playing online game and getting components from broadcast site.**

***Index Terms*— website automation, web-component analysis, website behavior analysis, website integration**

## I. PREFACE

Nowadays internet applications are widespread. It is declared that the era of Web 2.0 technologies came up – that means the idea of the proliferation of interconnectivity and interactivity of web-delivered content [1]. Sites incorporate new technologies of creating dynamic web pages and complicated user interaction scenarios. There are various technologies of creating dynamic content and it is not so easy to unite them or use some part of one site in the other as it was in pure HTML –era. As content is generated at the server side, it is not enough to send URL as request and analyze the reply of the server, as it is just a static stamp of some particular case.

As Web becomes some media for applications, the problem of non-transparency occurs. Yesterday one could just analyze the source code of webpage and understand the behavior – now it is pretty hard and non-trivial, some parts comes as black –boxes (for example flash container)

There exist tools that help to understand site structure, for example, Firebug for Mozilla but this tool does not allow automating information retrieval. The search robots distinct text and links and not components or behavior. There are cases when automation of using site is needed

Use some component from third-party site in your own

Automate actions on third-party site (post blog, play online game, etc.)

These tasks are usually solved locally that means writing a specific script for specific task. An example is various Greasemonkey scripts. In the paper a method of reconstructing site components, their behavior and interconnectivity is suggested so that then it will be possible to provide a visual environment of creating automation scenarios and solve the problem of components' reuse. The general case of playing online game is considered as a model of any site usage. The model is robust as it has three components - the user actions, the site logic, and other parties' actions.

Some difficulties to be faced

The plain HTML pages dissolve in the sea of the dynamic content namely PHP, ASP, Flash and others. The AJAX technology is also frequently used. One can name lots of other technologies used in creating web pages but the idea is that client –server communication is not based solely on hyperlink clicking but on different other scenarios. It results in a dynamically updated webpage. Dynamic content is generated either on a server side, e.g. PHP or on the client side e.g. JavaScript. For the moment let us neglect the problem of indexing such dynamic pages - that problem the Google successfully solves, but let us face other problem of linking different web applications. That seems easy if you want to just take some part of the webpage and encapsulate it in your own. Nevertheless, you can expect some difficulties with AJAX components and find out that as you moved a component it does not work properly for example the authorization procedure is missing. You have to reverse the code, which is trivial for static html and non-trivial for dynamic content and restore the communication model of the component you use.

Another difficulty exists if you want to automate information retrieval from the dynamic webpage. You may need to authorize or to navigate through some links, as the static link not always exists. That difficulty is now faced by search engines - the search robot indexes page but then the user follows the link provided and finds out other content. The solution used is to provide the webpage cached by search bot. This is not the best solution as it makes a security hole as one can have access the information on a particular webpage without required authorization and may have some copy legacy aspects. The search bot indexed the page under some account and cached page. That unintentionally reveals the content of authorized user to unauthorized user or user blocked by IP-filter.

As it seems trivial at the first sight to get some content from a webpage for your own use - it is not, so if the webpage is complex, dynamic, and recent technologies make it even worse. Usually some scripts for particular web content and particular site are written and locally solve the problem of retrieving information.

Not so long ago interactive web-applications were developed. They require the user to stay online for a significant time and provide dynamically changed online content. The example is online games - numerous of them. One of the most well known is Travian. It has more than 1000000 players worldwide and about one-third online at any moment. Another example is stock applications and online broadcasts. These services sometimes provide tools for automation and sometimes not, for example, online broadcasts are not recorded. If it is a big project, then adding some feature by its developers can take a long time. In addition, for any third party is it not easy due to the complexity of webpage structure.

It will be nice if we have a tool that can use for dealing with complex dynamic content – so we can distinguish the dynamic components, analyze them in a simple way and use them for our own needs. For example, we would like to have a bot that will play online game or record interesting broadcasts (the criteria can be based on number of active viewers) or automatically place bets on an auction when it is near the end.

## II. A GAME AS A GENERAL MODEL FOR A SITE

Let us construct a natural model of a game. In general, we can describe a strategy game as follows:

Each player initially has some number of resources R (let us say an army is a resource) and buildings B which provide resources in time.

A player can construct buildings by spending resources but new buildings provide resources faster

Resources can be exchanged one for other. Resources can be exchanged between players, so the war is considered as a resource exchange

There exist a set of actions that user can perform, that affect the system state

A simple model we can use is a model with feedback propagating in time

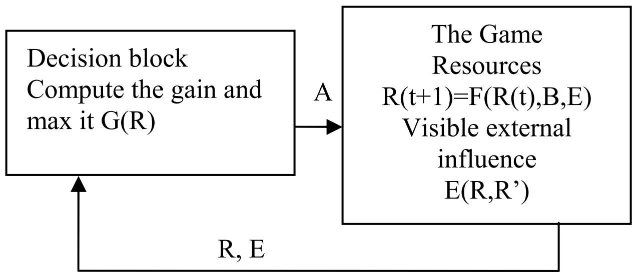


Figure 1. A simple model of the game

E indicates interaction with other players' actions. As we defined that interaction involves only resource exchange so we can say that given R(t) and R'(t) at the time t and applying E we get some resource flow $\Delta R(t+\Delta t)=E(R(t),R'(t))$ and $\Delta R'(t+\Delta t) = E(R(t),R'(t))$ respectfully. In fact we can consider the Nash equilibrium to be reached –so E is predictable and there exist a number of strategies that maximize the gain function. That means the function A(R,E,G) is defined for a chosen strategy G. F is a function that determines resource growth and can be derived from game rules with respect to R and B. As E can be determined either in a solid or in a probabilistic form then F can be determined as well.

That model can be spread on other applications for example in recording online broadcasts the resources will be the number of peers watching, the time of broadcast, disk space.

For example, we would like to watch the most interesting moments – then we should record only most viewed broadcast. (Suppose we record from screen and have a limit of one broadcast a moment)

Why should we use this model? The reason is that we want to deal with dynamic content in a nice way - have nearly all useful properties of the component accessible and automate our actions also in a nice way – just specify rules of the game or even automate learning the rules.

Analyzing communication protocol and modeling communication is considered in [2, 3]. There a finite state machine model is proposed to analyze the behavior and it works well if number of states is finite. In general we can consider states finite but there is a problem of describing such states as initially we do not know the component's algorithm and our goal is not model checking but finding out some relations between actions and resources in time.

So we want an analyzer, which on the input get some URL and maybe some rules (function G(R)) and on the output provide us with analyzed dynamic components and maybe some further rules of intercommunication (F, E)

## III. COMPONENT ANALYSIS

First, we need to specify control elements. For that purpose, a simple strategy can be implemented. A sniffer, browser and a program-controlled mouse is needed. Let us consider an action. First, we stay calm and do nothing and record some number of system states. We use screen version of browser to determine if anything changed and detect the regions where picture changed. That way we determine the "noise" generated by counters, banners. Then we try to check points on screen by moving mouse to the point and clicking. Not many points must be checked as the design is user-friendly and controls have reasonable dimensions. If the click was successful –that means something changed on a screen except "noise" then we try to specify the area that changed and sniff packets at the same time. Absence of packets may tell us that the control is a JavaScript and we should iterate through some more so we click around the area that changed or try to input some text – that way we determine the behavior of the control. Presence and type of packets can also give us information about control usage. Next, we have to try to map control into code. That can be done by using Firebug script for Firefox.

Then we should determine the static and dynamic components that are not controls and distinguish parts of the webpage that change by applying a simple differential scheme. That can be done using any client-imitating environment. The environment must fully support HTTP protocol and be able to track AJAX and JavaScript. The easiest way is to implement a sniffer for packets tracking and use custom scripts to control the Mozilla browser. Mozilla is the best choice because it is open source and easy to integrate with. Some part of the mechanism can be written even in grease monkey plug-in using JavaScript. Therefore, we can distinguish the dynamic part.

As a result, a set of dynamic components is created, including controls. If by activating some control, we capture communication we name it a resource and store to the database the content. The content itself can also be static and dynamic – that can be easily checked by series of tests by activating the same control with all possible feasible values.

Every dynamic component even noise is marked as a resource. Resources must have values so we must assign some parameters to components. In general, it is array of strings. The component can change its image, size (shape), text. First, we try with the latter and analyze the corresponding piece of code with regular pattern of readable letters. A dictionary or some method of word recognition is applied at that stage. If the procedure fails, we try to determine size change by differential analysis of series of images of the component and calculating overall estimate. If that approach also fail then we consider set of images and enumerate them in some way.

## IV. BEHAVIOR ANALYSIS

We perform a series of tests and try to find out restrictions and behavior. The user specifies the gain function and the model to use. While testing the gain is maximized by the following strategy:

Do nothing for some time and see what changes. Compute gain and make decision: if nothing changes or gain decreases –do some action, if gain increases do nothing for some time.

Do some action. Compute gain. Wait. Compute change in time and make decision: if nothing changes or gain decreases –do some action, if gain increases do nothing for some time.

In fact the idea is nearly to be Monte-Carlo method but with some little improvements. As a result, we obtain parameters of the behavior model of the site and a strategy of maximizing gain function.

There is still a problem of choosing appropriate model and it will be a further research which ones are better.

## V. EXPERIMENTAL RESULTS

The idea was implemented in a computer program. The main features of the program are the following:

- Separate static and dynamic content
- Use only natural user environment – that means HTTP protocol
- Have some feedback mechanism
- In dynamic content distinguish resources and action controls
- Provide some mathematical models for modeling function F
- Determine parameters of the model in a test series.

The tests were provided for the online game Travian. The account was manually created. Then the program worked. First, it used sniffer and applied the differential analysis to HTTP content, so resources were determined. Then by automatic browser, control the program clicked on the webpage and analyze if anything changed except the previously determined resources or if we went to other page by hyperlink. If something changed but we did not go to other page, it tried to see if some data can be input first by analyzing source code and if failed, by trying to “click and input”. The number of manipulations (clicks and inputs) was limited by a parameter. The “successful” combination meant resources change or page change and was recordered in database. Grouping was made by the webpage position of the component. The initial step was 10 pixels (between clickable points) and if a successful combination was found for some point, the area of component was determined by repeating the same actions for points nearby. The program distinguished resources components and determined the dependency of resources from time and actions, so that automated building was easily available. The program was also tested at the site smotri.com where the broadcast components and broadcast indexes were distinguished so it made available to construct a page similar to smotri.com broadcast page but without advertisement. Many problems occur in analyzing complex structures, model fitting, and war analyzing -that will be the topic of further research and improvements.

## VI. CONCLUSIONS

As web becomes more and more sophisticated, the problem of analyzing sites at the functional level will grow. The analysis at the user level provides large flexibility and transparency. A game model is applicable to some even non-game sites. A suggested concept of analyzing components and behavior naturally and empirically proved its value and however many problems emerge they are solvable and need further research.

## REFERENCES

[1] http://en.wikipedia.org/wiki/Web_2.0
[2] Analyzing conversations of web services T Bultan, X Fu et. al. IEEE Internet Computing 2006
[3] Jyotishman Pathak, Samik Basu et al. On Context-Sensitive Substitutability of Web Services In 5th IEEE International Conference on Web Services -2007
[4] ANALYZING USER BEHAVIOR On The Web, Eelco Herder, Ph.D. Thesis, University of Twente, 2006
[5] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.

# Software package for optimizing digital circuits[1]

Maxim Gromov, Natalya Kushik

***Abstract*—In this paper, we describe a software tool for optimizing the path length from primary inputs to primary outputs as well as the number of gates in digital circuits. A frame for optimization is extracted from a digital circuit; the extracted frame is divided into two parts and the maximum flexibility for each part is determined by the largest solution to an appropriate FSM equation. We check whether one or some output functions of each part can be replaced by a simple function of two primary input variables that can be implemented as a single gate, while preserving the behavior of the overall fragment. A developed software package can deal with digital circuits which have around 500 gates, and 40 primary inputs and outputs. Experiments were performed for a pack of benchmarks that were first resynthesised by ABC tool [1]. Our results show that the developed package can improve around 15% of benchmarks.**

***Index Terms*—Digital circuit, FSM equation, optimization**

## I. Introduction

THE complexity of digital circuits increases quickly and there still are no tools which can guarantee the design of an optimal circuit. For this reason, usually optimization tools run for already designed circuit. There are a number of optimization criteria such as reliability, fault-tolerance, minimal number of communication lines, delay, area etc. The problem of optimal design remains a challenging problem for developing new information technologies. The best approach for the optimization has been shown to be an iterative component optimization that can be based on solving an appropriate Finite State Machine (FSM) equation. A largest solution, i.e. the solution with maximum flexibility can be viewed as a reservoir for all possible optimizations of a frame of interest, from which an optimal frame implementation can be chosen. However, the complexity of solving an FSM equation generally is exponential in the number of states of its coefficients (FSMs). For this reason, a so-called window approach for optimizing digital circuits is used for optimization [2]. We iteratively extract a frame of an appropriate size from a given digital circuit, divide it into two parts and optimize these parts with respect to the given criteria. The procedure terminates when we are satisfied with the optimization results.

In this paper, when optimizing a circuit, we extract a combinational frame and then divide it into two component circuits (head and tail components) and optimize them based on the idea that in general, a number of combinational circuits can replace a head (or a tail) component without changing the behavior of the overall frame. All permissible replacements are represented as a nondeterministic circuit [3] that is derived as the largest solution to an appropriate FSM equation. For each primary output of a circuit component, we check whether the corresponding output function can be replaced by a simple function of two input variables. In this case, this function can be implemented by a single gate. Correspondingly, in the frame, length of some paths from primary inputs to primary outputs can be shortened as well as some gates can be deleted, i.e., there is a chance that the number of gates in the frame can be reduced.

The structure of the paper is as follows. Section II contains preliminaries. Section III is devoted to solving an equation for the head and the tail component. Section IV discusses software for optimizing digital circuits. Section V describes experimental results while Section VI concludes the paper.

## II. Preliminaries

In this paper, we use a *behavioral function* in order to represent a digital circuit behavior. For a combinational circuit the behavioral function $\Psi$ is defined over input and output variables of the circuit and $\Psi(\mathbf{x}, \mathbf{y}) = 1$ if and only if the circuit produces the output vector $\mathbf{y}$ to the input vector $\mathbf{x}$. Consider the combinational circuit in Figure 1.

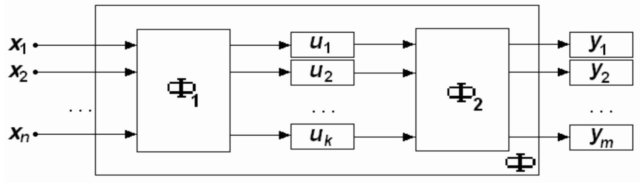


Figure 1. The combinational composition of two circuits

The circuit implements a system of Boolean functions $\Phi$ (an SBF $\Phi$) and can be described by a corresponding behavioral function $\Psi_\Phi(x_1, \dots, x_n, y_1, \dots, y_m)$: $\Psi_\Phi(X_1, \dots, X_n, Y_1, \dots, Y_m) = 1$ if and only if $Y_1 = \varphi_1(X_1, \dots, X_n), \dots, Y_m = \varphi_m(X_1, \dots, X_n)$. We

[1] This work is partly supported by RFBR grant N 07-08-12243 and by RFBR-NSC grant N 06-08-89500.

say that a function $\Psi$ is an SBF-behavioral function if $\Psi$ is a behavioral function of some system of Boolean, possibly non-deterministic, functions. Given a Boolean function $\theta$, we denote $M_{\theta}^{1}$ the set of variable values, for which the function equals 1. Given Boolean functions $\theta$ and $\Psi$ such that $M_{\theta}^{1} \subseteq M_{\psi}^{1}$, we denote this fact as $\theta \le \Psi$. The head component implements the SBF $\Phi_1$; the behavioral function $\Psi_{\Phi_1}$ of the head component is specified over the set $\{x_1, \ldots, x_n, u_1, \ldots, u_k\}$ of variables and we extend it over the set of variables $\{y_1, \ldots, y_m\}$. The tail component implements the SBF $\Phi_2$ and the behavioral function $\Psi_{\Phi_2}$ of the tail component is specified over the set $\{u_1, \ldots, u_k, y_1, \ldots, y_m\}$ of variables and we extend it over the set of variables $\{x_1, \ldots, x_n\}$. The behavioral function $\Psi_{\Phi}$ of the overall circuit which implements the SBF $\Phi = \Phi_2(\Phi_1)$ is specified over the set $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ of variables and $\Psi_{\Phi} = (\Psi_{\Phi_1} \wedge \Psi_{\Phi_2})_{\downarrow \mathbf{x},\mathbf{y}}$.

In order to optimize the head or the tail component of the frame we should replace a circuit component with another one preserving the external behavior of the composition. All such replacements are captured by a largest solution to a corresponding FSM equation. According to optimization criteria, an optimal circuit can be then extracted from a largest solution. In this paper, for each circuit component, we study whether it is possible to replace a component with another circuit which has less number of gates or has shorter paths from primary inputs to primary outputs.

## III. Solving an Equation over the Head Component and the Tail Component

### A. *Solving an equation over the head component*

The most flexibility for the head component can be captured by the largest solution to a corresponding FSM equation $\overline{(\Psi_{\Phi_2} \wedge \overline{\Psi_{\Phi}})_{\downarrow \mathbf{x},\mathbf{u}}}$, where $\Psi_{\Phi}$ is extended over the set $\{u_1, \ldots, u_k\}$ of variables and a digital circuit that implements the SBF $\Phi_3$ can replace the head component if and only if $\Psi_{\Phi_3} \le \overline{(\Psi_{\Phi_2} \wedge \overline{\Psi_{\Phi}})_{\downarrow \mathbf{x},\mathbf{u}}}$ [4], where $\overline{\varphi}$ is the inversion of the function $\varphi$.

The above statement gives a guide how to determine an SBF that can replace SBF $\Phi_1$ without changing the behavior of the overall system. We, thus, check whether one or more functions of the head component can be selected as functions of two input variables or as functions equal to the constant 1 (or to the constant 0) preserving all other functions. In this case, this output functions can be implemented by a single gate and all the gates of the path from inputs to a corresponding output which do not influence other output functions, can be deleted from the head component.

### B. *Solving an equation over the tail component*

The set of all permissible behaviors of the tail component can be captured by a partial FSM that is defined only for $u$-patterns which are output patterns of the head component. Thus, in order to get $u$-inputs where the behavior of the tail component cannot be changed we take the projection $(\Psi_{\Phi_1} \wedge \Psi_{\Phi_2})_{\downarrow \mathbf{u},\mathbf{y}}$. This function is not really a behavioral function, since it describes only a part of behavior. If for some $u$-pattern there is no $y$-pattern in the set $M_{\psi}^{1}$ then the behavior of the tail component for this $u$-pattern can be selected in an arbitrary way (so-called input don't care conditions). So we consider $(\Psi_{\Phi_1} \wedge \Psi_{\Phi_2})_{\downarrow \mathbf{u},\mathbf{y}}$ as a largest solution for the tail component and check whether there exits $y_i$, $i = 1, \ldots, m$, that can be replaced by a function of two input variables or by a function equal to the constant 1 or constant 0.

In our software we use Binary Decision Diagrams (BDD) [5] for all operators over Boolean functions. We use operators of the BDD package that is well known and is widely used when manipulating with digital circuits.

## IV. Software

In this section, we briefly describe the software package that is developed for optimizing digital, possibly sequential circuits. At the first step, a combinational frame up to 100 gates and 20-23 inputs is extracted. At the second step this frame iteratively is divided into two sequential parts which are optimized according to the above description and if the optimization occurs a component is replaced by a better implementation, the frame is divided again into two parts etc. The procedure terminates when we run out of time or are satisfied with the optimization results.

### A. *Circuit representation*

In our software package, we represent a sequential circuit given in the bench format as a set of connected gates with integer numbers. Each number uniquely identifies a gate. Correspondingly, the information of all gate predecessors (or successors) is represented by a Boolean matrix. The optimization process relies only on integer arrays: all the operations such as extracting a frame, optimizing a component, composing two circuits after optimization result also take place in integer arrays. Only at the last step this representation is back converted into the benchmark format (bench format). The use of such (hash) representation accelerates the optimization process compared with the representation where original strings of gate names are used without hashing.

When operating with behavioral functions BDDs are of a big help. All the operations such as deriving the behavioral function for a circuit, given in the bench format, deriving the largest solution, checking whether one or several output functions can be selected as constants (1 or 0), checking whether an output function can be a simple function of two input variables are performed fast enough for circuits which have up to 50 input and output variables. We use CUDD-package to calculate a largest solution as BDD for the tail component and transform it into a sum of products, as in this case, such representation seems to be more convenient than BDD representation.

### B. *Main methods of the software package*

**Frame extraction**. When extracting a frame we need to keep an eye on the correspondence between inputs and outputs of the extracted frame and gates of the initial circuit. We extract a frame without combinational loops and for this reason, we first order the combinational part of the initial circuit by layers depending on their distance from primary inputs and flip-flop outputs. If there are $n$ layers then we extract a frame as the set of all gates which belong to layers $j$, $j + 1, \ldots, k$, $1 \le j \le k \le n$.

**Deriving a behavioral function for a non-deterministic circuit that is the largest solution for the head component**. We use the BDD package in order to derive a behavioral function for each component. The largest solution then is obtained by BDD manipulation. Using the BDD representation of the largest solution each output function is checked whether it can be replaced by a constant or by a simple function of two input variables preserving the behavior of the overall composition.

**Deriving a behavioral function for a non-deterministic circuit that is the largest solution for the tail component.** For the tail component BDD representation of the largest solution is converted into a sum of products, as this representation seems to be more convenient for dealing with a system of partially specified Boolean functions.

**Optimization**. If one or several output functions of a head (or tail) component can be replaced by a constant or by a simple function of two input variables then a corresponding gate is added to the component and all gates of the initial component which do not influence other outputs are taken away.

**Insert operator** is used for inserting the optimized component into the frame and then for inserting the obtained frame into the initial circuit.

## V. Experimental results

We have conducted experiments using the proposed method with some benchmarks [6] in order to see how often our package can reduce the number of gates and the length of a path from primary inputs to primary outputs for a given combinational circuit. We used ABC for logic synthesis and verification. A given benchmark was first synthesized as a logical circuit using ABC and our package was used for the circuit optimization. Extracted frames have up to 25 inputs and path length from primary inputs to primary outputs varies from 5 to 19 being 10 on average. Ten functions of two variables, such as AND, OR, etc., were used for optimization; all of them can be easily implemented by a single gate. The results show that the developed package can improve around 15% of benchmarks. The optimization is not huge but on the other hand, those benchmarks were already optimized many times using other packages.

## VI. Conclusions

In this paper, we described the software tool for optimizing the number of gates in digital circuits as well as the path length from primary inputs to primary outputs. A combinational frame extracted from a digital circuit is divided into two components. Each component then is optimized independently. We experimented on some benchmarks from [6] and our results clearly show that there exist a number of benchmarks such as s838.bench, s298.bench and s420.bench, etc. for which our package returns optimized circuits. More experiments with new benchmarks are needed in order to estimate the efficiency of the developed package.

## References

[1] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, http://www.eecs.berkeley.edu/alanmi/abc/

[2] S.Zharikova, M.Vetrova, N.Yevtushenko Optimization of a multi component digital circuit by solving a system of FSM equations // Proceedings Euromicro Symposium on Digital System Design Architectures, Methods and Tools, IEEE Computer Society. – Belek-Antalya, Turkey, 2003, pp. 62-68.

[3] A. Mishchenko, R. Brayton, R. Jiang, T. Villa, and N. Yevtushenko, "Efficient solution of language equations using partitioned representations", Proc. DATE, 2005, pp. 412-417.

[4] N. Kushik, G. Sapunkov, S. Prokopenko, N. Yevtushenko. Minimizing path length in digital circuits based on equation solving. In Proc. of IEEE EAST-WEST design&test symposium, October, 2008, pp. 365-370.

[5] CUDD [Electronic resource] –http://vlsi.colorado.edu/~fabio/CUDD/

[6] Education: Virginia Tech: The Bradley Department of Electrical & Computer Engineering / College of Engineering; ISCAS89 Sequential Benchmark Circuits. – Access mode to an electronic resource. http://www.ece.vt.edu/mhsiao/iscas89.html is free.

# UDD network model for time-limited data storing

Scherbakov Konstantin
SPbSU
Saint Petersburg,
Russia
konstantin.scherbakov@gmail.com

## ABSTRACT

Different techniques of using peer to peer networks are mentioned. The process of data distribution in peer to peer network by the single node's initiative for purpose of further downloading by this node's owner is described. A common model of special peer to peer network (UDD network), developed for single node initiated data distribution, with its main characteristics and used algorithms is introduced. Some ways of improving existing peer to peer networks for the purpose of compliance to UDD network specification are mentioned. Experimental results of single node's initiated data distribution in one of the existing peer to peer networks (with our improvements applied) are analyzed.

## Introduction and related work

Pure and hybrid peer to peer (p2p) networks are widely used now in our life. There are at least two main ways to use them:

- for grid/cloud computing/storage (Amazon S3, Nirvanix SDN, Gridnut, Wuala etc.) [1,5,9,13,19,25]
- for data exchange between interconnected nodes (bittorrent, exeem, ed2k, kademlia, gnutella 1,2 etc) [2,4,6,7,21,22]

If p2p network is designed for data exchange, new nodes usually connect to it for the purpose of retrieving some files, they are interested in, or share to other nodes some files, they want to. But there is at least one more alternative way of using such type of p2p networks. One node can connect to network for storing its own data/files over other nodes for the purpose of retrieving this data by node's owner from any other place. This data may be uninteresting for other nodes of network and may be encrypted. So let's call such type of data the *u-data*. This way is rarely used because of lack of data access control, data availability control and u-data fast distribution mechanisms in existing p2p networks.

Our main goal is to introduce the architecture of prototype of such p2p network and search request routing/data storing/indexing protocols for it, that allows any node to store securely its u-data over the network for some fixed time interval and also grants this node's owner ability to retrieve his data from any other location during fixed period of time mentioned below and control availability of this data over the network during its limited storing period. Using of these mechanisms by particular node should depend of some coefficient of node's usefulness for the whole network. So nodes, that are more useful for the network, should have opportunity to use these mechanisms more often and more completely.

## UDD P2P network model

Let's assume that one person or group of persons have some u-data, that is very useful for this person or group of persons, but not interesting for other participants of our future u-data distribution (UDD) p2p network. This person wants to have access to this data from any place, where he can connect to internet (and to our UDD p2p network too). He also wants to have ability to extend his data storing time and he expects that there is some mechanism used in this network that prevents with high probability his data from being deleted from all network nodes during its limited storing time.

What properties we expect for our UDD p2p network? Let's assume that our network will have the properties listed below:

1. Network nodes should have their own limited long-term memory
2. Network nodes should have access to some computing resources (for example their own CPU) to calculate data content and keyword hashes
3. Network should have some mechanisms for u-data storing and effective distributing and deleting
4. Network should provide high probability data disappearance preventing mechanism
5. Network should have mechanisms of effective data search request routing and metadata storing [3,8,10-12,14,15,17,20,23,24,26,27]
6. Network nodes may connect and disconnect from network constantly and therefore network provide some mechanisms of retrieval node connect/disconnect rate

This is ideal network for our purposes. Real p2p networks (especially file sharing) often have 3-4 of these properties, but we won't examine them at this point. It's only important to note, that some of this networks may be rather easily extended in special way to have all properties of our ideal p2p u-data storing network [16]. It's important to introduce and describe the method of u-data distribution in UDD network initiated by some specific node.

In usual p2p network data distribution process usually have some general stages:

- Data hashing (data content hashing, data description keyword extraction and hashing etc.)
- Data announcing (more precisely this stage should be called "data hashes announcing")
- Waiting for external requests for announced data from other network nodes
- Data uploading to external network nodes interesting in data announced

It's a good way of distribution process for data that may be interesting and useful for other peers in usual file sharing p2p network, where waiting for external requests stage hasn't infinite durability for the data being distributed. But we assumed that our data is uninteresting for other nodes. Furthermore, our data may be encrypted by its owner to prevent effective using of it by other network node's owners. Therefore 3rd stage of usual data distribution process will have infinite durability for u-data in usual file sharing network. So how we can effectively distribute data in our ideal network? Nodes have some limited long-term memory. If we try to send our data in all accessible nodes, some of them may reject it, because of lack of free disc space. Our first goal is to share u-data between N nodes (including the initial one), where N is defined by data and initial node's (n_init) owner (N should be limited according to the node n_init usefulness coefficient, mentioned above). Lets assume that n_init have M neighbors. If M>N we will ask them about their free space and ask them to reserve *sizeof(u-data)* bytes for our u-data storing with M queries and get M_ok results with satisfying answer. If M_ok < N, we will ask this nodes for their neighbor lists and then repeat our first step. If M<N, we can ask them for neighbors first and then ask extended node list for free space reservation. But we can save some network bandwidth by uniting these two types of requests into a single one. So n_init can ask its neighbors to reserve some free space for its u-data and in the same request it can ask them to forward this request to their neighbors. After receiving at least M satisfying results, n_init sends to each of them full list of nodes from this list and offers to receive his data. M nodes start to receive this data, using swarm cooperation. After that they receives a unique id's, which are actually a structure with some fields: data hash, data lifetime/finish storing date, replication coordinator priority/id (this value will help to determine current replication coordinator for stored u-data), M, owner node's data access key (a special access key for this portion of data only), owner node's id key, search keywords, etc. This process is briefly described in the code listing below

Listing 1:

```
$neighbors_arr = get_neighbors(); //neighbor list
$m_ok=0;
foreach ($neighbors_arr as
$neighbor_num=>$neighbor_value)
{
 if (reservespacerequest(
 &$neighbors_arr[$neighbor_num],&$m_ok,
 $space, $datahash)==1)
 $neighbors_arr[$neighbor_num]
 ['reserved']=1;
 $neighbors_arr[$neighbor_num]
 ['checked']=1;
}
$cur_neighbor= get_firstkey(&$neighbors_arr);
while ($m_ok<$n)
{
 addneighbors_callback(&$neighbors_arr,
 &$m_ok,$cur_neighbor,
 'reservespacerequest');
 if(($cur_neighbor=get_nextkey(
 &$neighbors_arr,$cur_neighbor))===
 false) break;
}
$invited=send_download_invitation( &$neighbors_arr);
/*some code deleted from listing*/
$resultnodes=send_data_info( &$neighbors_arr);
```

As was mentioned above our ideal network can be constructed by extending some real p2p file sharing network (for example bittorrent of ed2k/kademlia network). But if we realize a possibility of distributing u-data in some client application for these networks without any limitation, very few nodes will use it, because while being unlimited, this process can make a lot of parasitic traffic in the network and slow down downloading and uploading of usual files. So if we want to extend these networks with u-data distribution ability, we need to describe some methods that can guarantee limited use of this function, for example by useful for whole network nodes only. Usually coefficient of usefulness is represented by formula

*coef_usf=node_upl/node_dwn,*

where node_down>0, and infinity otherwise.

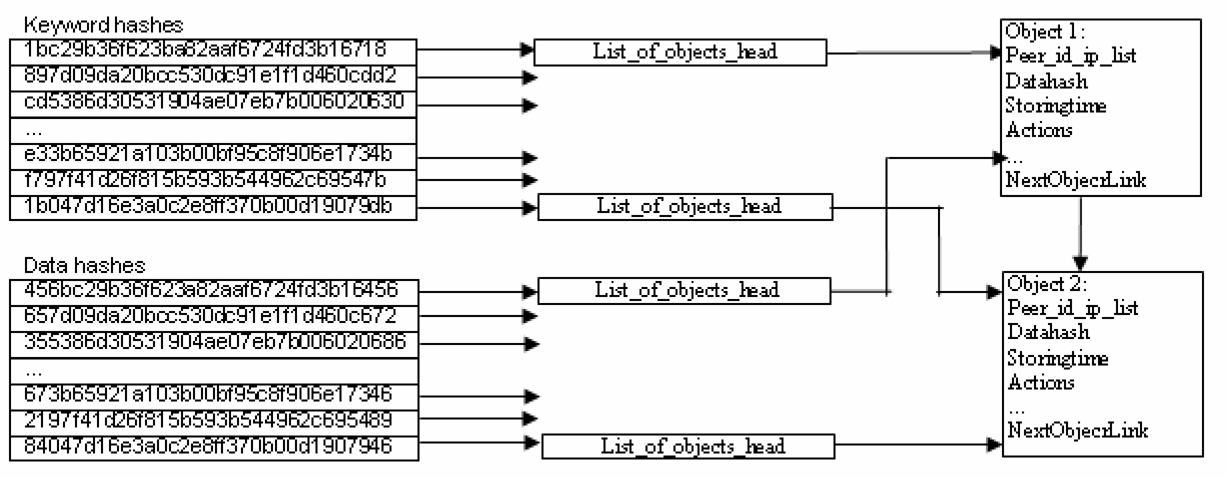
When any nodes download some data they need, their node_dwn values increases, and when they upload some data to other nodes, their node_upl values increases. These values can never decrease. For our extended network, we can change this formula to

*fair_coef_usf = (u_usual+ deltads*d_special) / (d_usual + deltaus*u_special),*

where u_usual – outgoing traffic value for usual data, d_usual – incoming traffic value for usual data, u_special and d_special – incoming and outcoming traffic value for special data (distributed over our u-data distribution mechanism), deltads, deltaus – special weight coefficients, introduced for correcting value of fair_coef_usf after using by some node possibility to distribute u-data .

Now let's return to the end of u-data distribution process. Let's assume that our network have some special nodes, that allow storing data indices (it's true, if we'll extent bittorrent or ed2k network, that are now).

All M nodes with our data stored on it sends data hash, finish storing date, search keywords to their neighbor nodes, which are marked as indexing nodes, which


Keyword hashes
1bc29b36f623ba82aaf6724fd3b16718
897d09da20bcc530dc91e1f1d460cdd2
cd5386d30531904ae07eb7b006020630
...
e33b65921a103b00bf95c8f906e1734b
f797f41d26f815b593b544962c69547b
1b047d16e3a0c2e8ff370b00d19079db
Data hashes
456bc29b36f623a82aaf6724fd3b16456
657d09da20bcc530dc91e1f1d460c672
355386d30531904ae07eb7b006020686
...
673b65921a103b00bf95c8f906e17346
2197f41d26f815b593b544962c695489
84047d16e3a0c2e8ff370b00d1907946
List_of_objects_head
List_of_objects_head
List_of_objects_head
List_of_objects_head
Object 1:
Peer_id_ip_list
Datahash
Storingtime
Actions
...
NextObjectLink
Object 2:
Peer_id_ip_list
Datahash
Storingtime
Actions
...
NextObjectLink

stores these values in a special data structure, you can see on the scheme 1 below.

As we can see on this scheme, our special node has 2 hash tables. First hash table contains hashes of search keywords, stored in dynamic array as array keys. Each value in this array contains link to a special list of pointers to objects, where each object contains content hash for data, relevant to this keyword, data note and filename, node id's or addresses, where this data is stored. We can store ids for nodes with dynamic ip addresses and raw ip addresses for nodes with the static ones. But if we only have hash table for keywords, our special nodes will be useless for search by data content hash request routing. So we'll create a second hash table for this purpose. It contains known data content hashes links to special node, with links to the head of special object list, which also contains links to real objects with data about data hash, keyword hash, node id's and/or ips, mentioned above. So, when search request arrives, we split it to keywords and search for their hashes in first table. Then we get data filenames, keyword relevance in some way, data notes and send it back to node, that initiated this request and let it decide, what to do with the result retrieved. When hash search request arrives, we send back to the request initiator ids / ip addresses of nodes that can store this data.

How we can use this structure for updating storing-time of u-data distributed in our network? Node n_init should send special request with data hash and new storing time to all known nodes with our hash table structures. Then these special nodes should find a record about this data by its content hash. And finally they should update storing time for this record and send update request to known nodes with this data stored. This is non-guarantied way to update data storing time, but our goal is to update data on some nodes. All other work should do current data availability coordinator.

Listing 2:

```
$reqtype=get_rec_type($request);
if ($rectype=='updatetime' &&
($newtime>time()+$delay))
{
  $dataobj=0;
  search_by_datahash($hashstructure2,
  $hash,&$dataobj);
  update_storingtime_delayed(&$dataonj,
  $newtime,$delay);
  mark_for_update(&$dataonj);
  send_updatetimerec_delayed(&$dataonj,
  $newtime);
}
elseif ($rectype=='deleterec')
{
  $dataobj=0;
  search_by_datahash(&$hashstructure2,
  $hash,&$dataobj);
  send_deleterec_delayed(&$dataonj,
  $delay);
  delete_delayed(&$dataonj,$delay);
}
/* $delay represents time in seconds, after which this object is
actually deleted after it was disabled for search requests */
```

In the listing 2 we can see code that allows indexing nodes to update or delete objects with data hashes and nodes ids/ips lists. It's significant that objects are not deleted immediately, so we only mark them as deleted and set some delay time, after that it will be actually deleted during the regular indexing node's maintenance process. This process should take place regularly in the periods of low CPU/network/etc load of indexing node. While maintenance, indexing node should check all objects for it's data storing time and if necessary, physically delete them, if undelete flag is not set. Else if this flag is set, node indexing should take away delete flag and update storing time, if it's higher than current time, and if there is no delete flag for this or higher storing time for this data hash, else node should also delete object permanently.

So, how we can use this structure for searching u-data, distributed over the network and for updating its storing time / non-guaranteed deleting etc.?

Here you can see this process:

Listing 3:

```
if (is_server_load_low())
{
  foreach ($hashstructure2 as $hkey =>
  $hvalue)
  {
    $actionlist=getactions(
    &$hashstructure2[$hkey]);
    if ($actionlist)
      writelog(executeactions(
      &$hashstructure2[$hkey],
      $actionlist));
    while (!is_server_load_low())
    {
      sleep(5);
    }
  }
}
```

We have mentioned data coordinator below. Let's briefly describe its functions. Regularly all nodes, storing our u-data sends requests to current coordinator to check its availability, tell it that they are available and receive a list of other nodes with u-data stored. Coordinator monitors these requests and has a list of currently active nodes with u-data presented. If coordinator suddenly disconnects, other nodes elects a new one by special data id, they have. They use last received nodes list in this process. Node with minimum data id (this may be UNIX time for example) wins the election and became a new coordinator. When the old coordinator arrives back in network, he sends requests to known nodes with u-data stored and receives new coordinator address. Then he tells new coordinator to tell other nodes about new old coordinator send current node list and became a regular node.

## Experimental results

And now let's make some experiments with our data distribution mechanism in the real network. We have extended existing bittorrent client bittornado, torrent tracker tbdev [2,21,22] and a special tool to emulate large number of different nodes (this tool is written on PHP). We are using 2 servers: C2D E4400, 3GB ram, Centos 5.0 and C2D E6400, 3GB ram, FC6. We know some statistics for the normal bittorrent network, based on this 2 servers: network average size = 4100 peers; about 1700 peers connected to network with intervals more 24 hrs. Average peer renewal speed: 0.08 peer per second. Average incoming/outcoming speed of all peers = 1.4 mbit/s

Nobody other then us was given modified p2p network client, so we will create virtual nodes on server 1, make server 2 indexing node and start to distribute data over our virtual network with most characteristics equal to the real one.

Firstly we'll assume, that all peers have appropriate disc space for our data and we wont's actually save it on it's discs. We'll only save data hashes, storing time and indexes instead of it. 1700/4100 * 100% roughly = 41%. So in all our experiments 41% of peers will be always connected to network.

In first two sets of our experiments we'll set and fix the coordinators updating interval to 1 hour and start to distribute data to varying number of nodes. We'll also vary average number of connected nodes (in experiment set two). Our goal is to determine conditions, when last node with our data will leave network and it became inaccessible and how many nodes we need to have 25% of nodes at the end of one coordinator update period.

Then we'll also vary coordinator updating interval and will determine the same conditions, when all stored data will disappear. But firstly we'll assume that our network doesn't have a constantly connected (core) nodes.

Every experiment we'll repeat 100 times to determine best and worst results for current conditions.

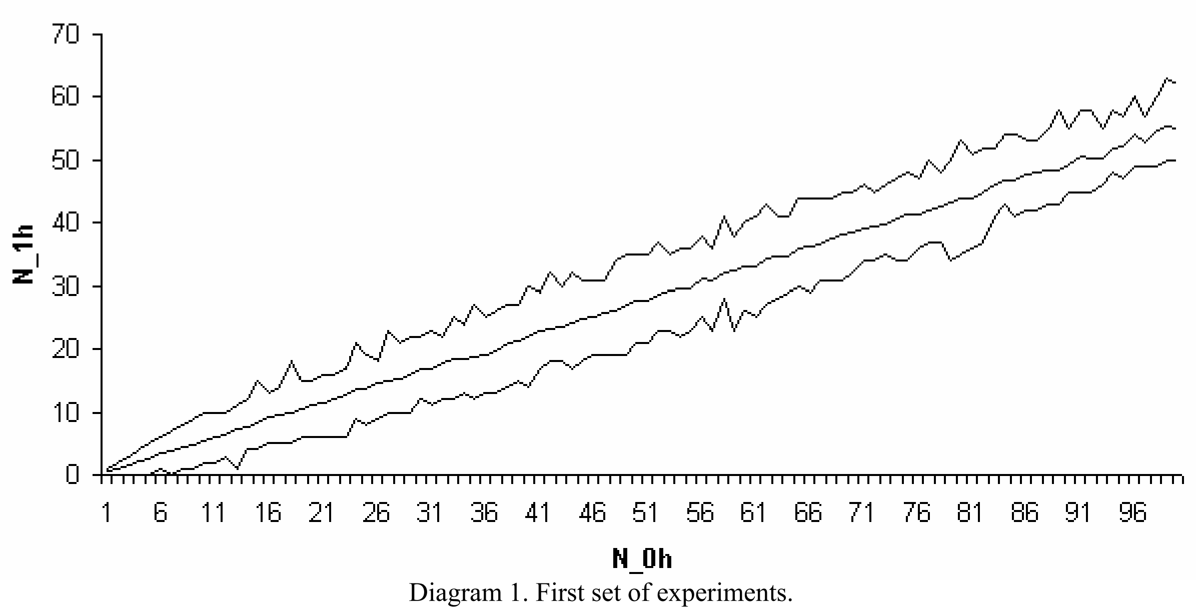


Diagram 1. First set of experiments.

In the first set of experiments (100 nodes connected to network in average, coordinator update time = 1 hour; every hour 59 random nodes leaves network and 59 other connects)[1] we can see, that when N (total nodes to

[1] Raw data for this set of experiments can be found at http://195.70.211.9/syrcose09_set1.txt

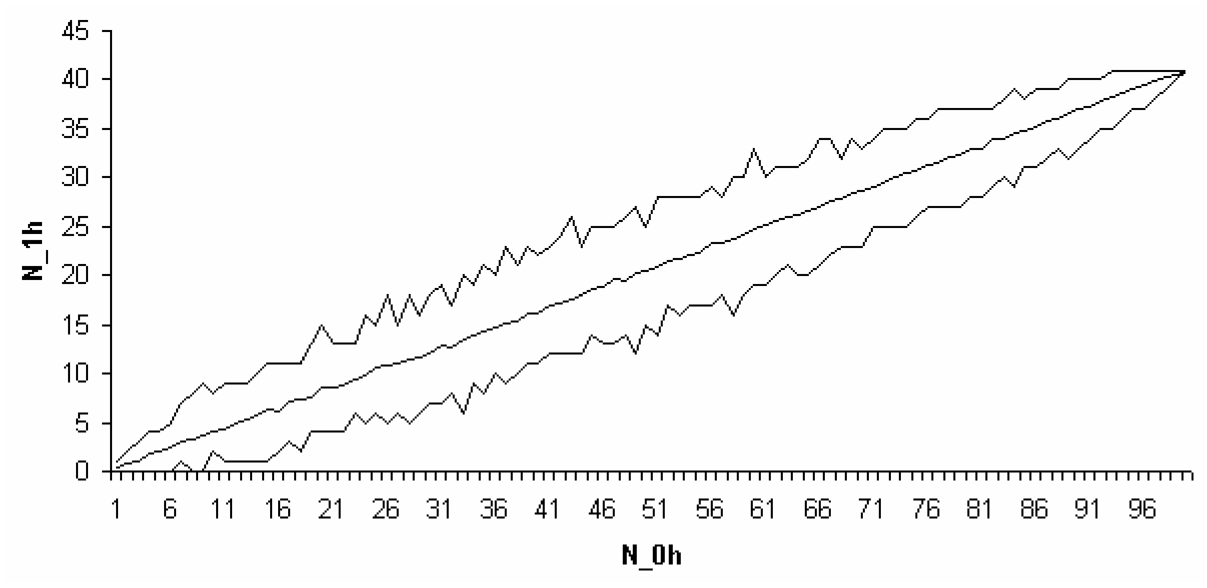


Diagram 2. Second set of experiments.

which our data is distributed initially) reaches value of 14, more than 25% of nodes will still have our data on it after 1 hour in the worst result acquired. Theoretically N should reach the value of 60 to give as a guarantee of data saving after one coordinator update period at least on one node, but practical results are better, because probability of the event "all nodes with data leave our network after one coordinator update period" decreases exponentially (and will be about O(10^(-59*2)) for N=59 ). So in real network we don't need to distribute data on such huge amount of nodes to save our data with probability very close to 1.

When taking a look to diagram 2 (second set of experiments, where we have 41 persistently connected nodes, and other properties are equal to set 1)[2] we can see slightly higher number of minimum required nodes for data saving on 25% of nodes at the end of coordinator update period. It's close to 25 nodes, because in this set of experiments 59 unique nodes leaves the network, and in set 1 this value is distributed in the interval (1,59).

Now let's vary total number of connected nodes with other properties of network equal to set 2. At the Diagram 3, which represents the 3rd set of experiments[3] we can see results for 4000 connected nodes and for N from 1 to 2996 with step 85. While N reaches value of 86, we have more than 50% of nodes with our data alive after coordinator update period

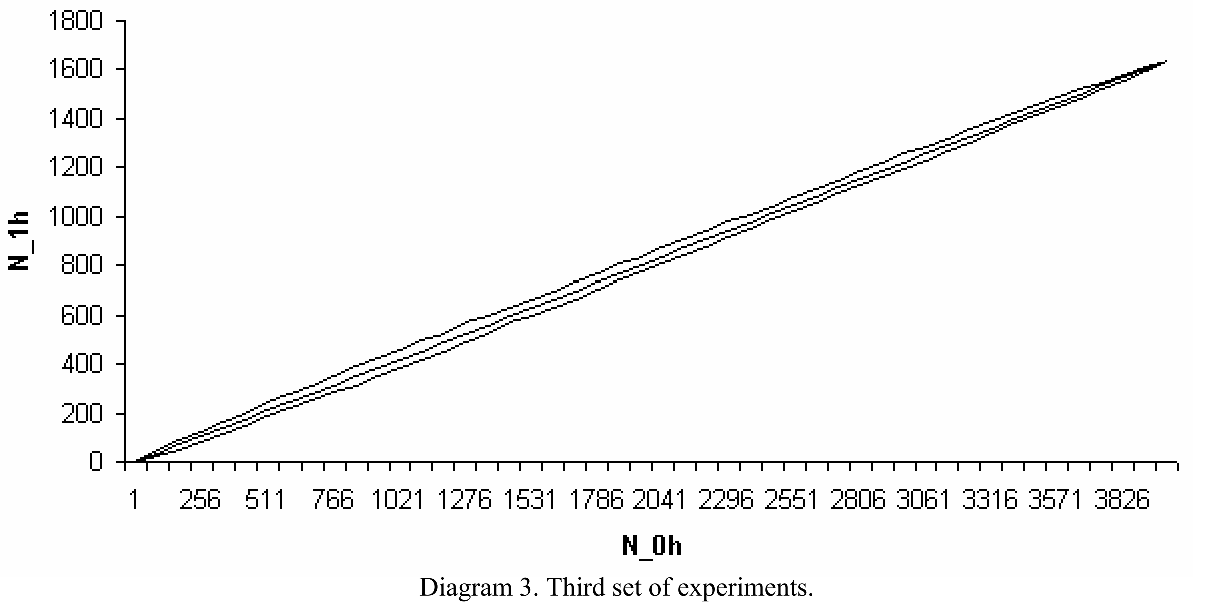


Diagram 3. Third set of experiments.

[2] Raw data for this set of experiments can be found at http://195.70.211.9/syrcose09_set2.txt

[3] Raw data for this set of experiments can be found at http://195.70.211.9/syrcose09_set3.txt

While network connect/disconnect rate is fixed, it's better to have more non-persistently connected nodes for minimizing the number of nodes, to which our data should be distributed for having the probability of saving very close to 1 after one coordinator update period. This probability will decrease with time very slowly, so we'll have its high enough at the end of our data storing interval, because it's not equal to infinity. More experiments are required to determine maximum satisfying storing time interval for high probability of data saving. We can also say that varying coordinator update interval can help us to increase the probability of data saving. It's important to rightly determine update interval before distributing any data in network and vary this interval while data life cycle to minimize the number of nodes to which data is distributed and save the network bandwidth and node's computing resources. We can do this by sending special requests to indexing nodes for example (we have one of them in our experiments – it's an extended bittorrent tracker). These actions will make a little additional non data-transfer traffic over the network, but will save us significantly more traffic between nodes.

## Conclusions

So, we have described a model of ideal p2p network for data distribution initiated by a single node. We also introduced some methods, that should be used in such type of network and make three series of experiments with good results. But this work is unfinished now. Our next work will be concerned to examining more deeply routing mechanisms in that type of networks and introducing methods that will allow making data distribution and retrieval more secure.

## REFERENCES

[1] Amazon S3 official documentation: http://docs.amazonwebservices.com/AmazonS3/2006-03-01/

[2] BitTorrent full specification (version 1.0). http://wiki.theory.org/BitTorrentSpecification

[3] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In ICDCS, July 2002.

[4] Exeem project specification: http://www.exeem.it.

[5] I. Foster. Peer to Peer & Grid Computing. Talk at Internet2 Peer to Peer Workshop, January 30, 2002

[6] Gnutella project specification: http://www.gnutella.com.

[7] Kademlia: A Design Specification. http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[8] V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In CIKM, 2002.

[9] J. Kubiatowicz, D. Bindel, Y. Chen. Ocean-store: An architecture for global-scale persistent storage. In ASPLOS, 2000.

[10] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In ICS, 2002.

[11] D. Menasce and L. Kanchanapalli. Probabilistic Scalable P2P Resource Location Services. SIGMETRICS Perf. Eval. Review, 2002.

[12] I.Nekrestyanov. Distributed search in topic-oriented document collections. In SCI'99, volume 4, pages 377-383, Orlando, Florida, USA, August 1999.

[13] Nirvanix SDN official documentation: http://nirvanix.com/sdn.aspx

[14] S. Ratnasamy, P. Francis, M. Handley. A scalable content-addressable network. In ACM SIGCOMM, August 2001.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Middleware, 2001.

[16] K. Scherbakov. Search request routing in Bittorrent and other P2P based file sharing networks, SYRCoDIS 2008, Saint-Petersburg, Russia

[17] I. Stoica, R. Morris, D. Karger. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM SIGCOMM, 2001.

[18] M. Stokes. Gnutella2 Specifications Part One. http://gnutella2.com/gnutella2_search.htm.

[19] D. Talia, P. Trunfio. A P2P Grid Services-Based Protocol: Design and Evaluation. Euro-Par 2004

[20] A. S. Tanenbaum. Computer Networks. Pren-tice Hall, 1996.

[21] TBSource official documentation: http://www.tb-source.info

[22] TorrentPier official documentation: http://torrentpier.info

[23] D. Tsoumakos and N. Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. In 3rd IEEE Int-l Conference on P2P Computing, 2003.

[24] D. Tsoumakos, N. Roussopoulos. Analysis and comparison of P2P search methods. Proceedings of the 1st international conference on Scalable information systems, Hong Kong, 2006

[25] Wuala project official documentation: http://www.wuala.com/en/about/

[26] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In ICDCS, 2002.

[27] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location androuting. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001

# Robust Short Message Protocol

Vitaly U. Klimashov, *RSFLabs*

**Nowadays, the software and hardware solutions based on Linux OS and open-source software became popular not only for server or desktop use; they are also widely employed by the developers of the distributed control and telemetry systems, including the mission critical ones. The goal of our research is design and implementation of a reliable point-to-point message transmission protocol.**

**ISO – International Organization for Standardization,**
**RSMP – Robust Short Message Protocol,**
**RTT – Round-Trip delay Time,**
**TCP – Transmission Control Protocol,**
**UDP – User Datagram Protocol**

## I. Introduction

Modern computer communication systems are crucial for different kinds of military, government and civil applications. Evergrowing demand for automated control and management systems results in growing requirements; especially strict are the robustness requirements for the communication systems that operate in harsh conditions and under heavy load. The same requirements apply for many other different kinds of applications and systems that rely heavily on the data transfer layer. In a modern dynamic world where the computer networks appear and disappear fast, the task of their interconnection is especially important, as well as the common standardized interprocess communication protocols.

Some large commercial organizations develop their own protocols to suit their specific needs. Those protocols are often designed to be used with certain kinds of hardware and software, so that their range of possible applications is narrowed severely.

## II. Problem Analysis

Our company had taken part in the design and implementation of a geographically distributed remote control system where the messages were transmitted over Internet. Some of the key requirements were: robustness, noise immunity, guaranteed data delivery and highest performance possible. At first it was decided to employ TCP for telemetry and remote control messages transmission because of its wide recognition and known robustness. But during the design phase of the project it turned out that most of the target nodes are located in the distant areas far from large settlements and hence out of the zone of reliable Internet access. The actual quality of the available data transmission channels was extremely poor: lost packet rate up to 30%, average transmission delay up to 300 ms. It became obvious that TCP didn't quite suit our needs. While providing robust, guaranteed data transfer, it actually had very poor performance because of large amount of auxiliary data being sent. As the delivery time was of our greatest priority, we were forced to cease using TCP. Such harsh working conditions forced us to investigate profoundly available data transmission protocols in order to find one that would suit our specific needs.

The most widespread protocols, such as TCP and UDP, are standardized by ISO. The developers are allowed to employ them freely whenever needed. At the same time, most of those protocols have quite strict limitations and thus are not suitable for use with low quality data transmission channels. Protocols governed by ISO are fundamental and generic by nature, and do not perform well under extreme conditions. In such cases, adaptation of the protocols is required.

## III. Proposed Solution And Current Status

In order to solve the problem of reliable data transmission over low-quality data transmission channels we had decided to develop a new protocol based on UDP, which is fast enough to suit our needs of building the real time distributed systems. We had to extend UDP with certain features in order to meet the requirements, i.e. guaranteed data delivery and automatic protocol self-adaptation upon channel quality changes.

The protocol is of peer-to-peer kind. All of the packets being sent are grouped into sessions. The packet structure is based on the UDP packet structure; the only difference is the additional 1-octet field that carries the sequence number of the packet. Such numbering allows for lost packet detection and retransmission on demand. Instead of sending every packet once, it was decided to use preventive retransmission: every packet is sent in a given number of copies called “a pack.” Sequence numbering and preventive retransmission are the basic functions used to achieve guaranteed data delivery. Packet structure is depicted at the figure 1.

Manuscript received March 26, 2009.

A. A. Emelyanov is an engineer of RSFLabs, Russia, N.Novgorod, 603104, Nartova, 6. phone: +7831-278-90-47 url: www.rsflabs.biz

R. M. Dmitrienko is an engineer of RSFLabs, Russia, N.Novgorod, 603104, Nartova, 6. phone: +7831-278-90-47 url: www.rsflabs.biz

V. U. Klimashov is an engineer of RSFLabs, Russia, N.Novgorod, 603104, Nartova, 6. phone: +7831-278-90-47 url: www.rsflabs.biz

V. V. Mokhin is an engineer of RSFLabs, Russia, N.Novgorod, 603104, Nartova, 6. phone: +7831-278-90-47 url: www.rsflabs.biz

|  |  |  |  |
|---|---|---|---|
| 0 Sender port 15 |  | 16 Receiver port 31 | UDP header |
| 32 Length 47 |  | 48 Checksum 63 |  |
| 64 Pack seq. No | 72 Payload |  |  |

Figure 1. RSMP packet

Auxiliary messages are also implemented. They perform connection establishment and release functions, as well as a few others (retransmission request, lost packs detection, etc). Unlike data messages, their payload is predefined: it consists of command codes and auxiliary data whenever needed. The structure of auxiliary messages is depicted at the figure 2.

|  |  |  |  |
|---|---|---|---|
| 0 Sender port 15 |  | 16 Receiver port 31 | UDP header |
| 32 Length 47 |  | 48 Checksum 63 |  |
| 64 0x00 71 | 72 Msg ID | 81 Auxiliary data |  |

Figure 2. Auxiliary RSMP packet

Data transfer itself is based on the UDP protocol. Hence, we've achieved relatively high operating speed if compared to TCP. At the same time, our extensions provide for guaranteed data delivery.

This protocol was named RSMP (Robust Short Message Protocol). It is intended to provide reliable data transfer service with the lowest delays possible over the low-quality data transfer channels with significant packet loss and RTT. It is supposed that the protocol should be employed for short telemetry and remote control messages transmission in the distributed monitoring and control systems.

Its primary advantages are:

- guaranteed data delivery;
- high performance;
- automatic self-adaptation upon channel quality changes

We have performed extensive testing of the implemented protocol. It was confirmed that the software developed is able to perform well in harsh conditions. The software was tested and is already used in the several parts of the geographically distributed telemetry and remote control systems located in Volgograd and Nizhny Novgorod. The protocol was proven to provide reliable service when used with low-quality data transmission channels with random delays and packet loss.

## IV. Future Prospect

The RSMP protocol is supposed to be employed widely as a part of distributed remote monitoring and control systems. Its features allow for building reliable telemetry and telecontrol systems even in the distant rural regions.

# Telecommunication protocols development, simulation and code generation tool

Roman M. Dmitrienko, *RSFLabs*

**The subject of our work is an original interactive telecommunication protocols design, simulation, and prototyping tool set. The following subjects were investigated during the research phase of the project:**

**- Efficient modeling of dynamic, unpredictable data structures,**

**- Reliable yet tunable code generation routines.**

**The idea of this presentation is to show modern efficient methods of design of efficient complex software solutions.**

**API – Application programming interface,**
**ASN.1 – Abstract Syntax Notation 1,**
**CASE – Computer-Aided Software Engineering,**
**DTD – Document Type Definition,**
**GCC – GNU Compilers Collection,**
**GPL – General Public License,**
**GUI – Graphical User Interface,**
**IPC – InterProcess Communication**
**LGPL – Lesser General Public License,**
**MSC – Message Sequence Chart,**
**OS – Operating System,**
**RTOS – Real Time Operating System,**
**SDL – Specification and Description Language,**
**SDL-RT – Specification and Description Language for real time applications,**
**XML – eXtensible Markup Language**

## I. Introduction

Our research and development of the software is conducted as a part of an internal corporate project aiming the rapid prototyping of custom telecommunication solutions. This project has two extremely important distinguishing factors:

- The systems under development are subject to continuous changes,
- The reliability of the systems being developed is much more important than their performance.

In order to cut the systems prototyping and development costs, it was decided to employ CASE rapid prototyping tools. The two aforementioned factors imposed certain restrictions and requirements on the software development process, tools and algorithms involved, thus stipulating the need of in-house CASE software development. As an additional cost-cutting measure, only open Linux-based tools and libraries were employed during the development phase.

Manuscript received March 20, 2009.

R. M. Dmitrienko is an engineer of RSFLabs, Russia, N.Novgorod, 603104, Nartova, 6. phone: +7831-278-90-47 url: www.rsflabs.biz

## II. Project Internals

The idea of the software, while quite complicated, is fairly straightforward. From the user's point of view, the standard workflow should include defining the system using SDL-RT [1] language, eliminating possible errors, running source code templates generation routine, as well as optionally simulating the work of the protocol and tracing its input/output, estimating the characteristics of the protocol, generating and running test scenarios. When all of the required phases are finished, the user should inspect and complete the generated source code templates as needed, deploy the system, run smoke tests etc. The former is achieved using the software we are developing; the latter is done by the user.

The major problems we met were:

- Efficient, reliable modeling of dynamic, constantly changing data structures during the system design phase,
- Entirely customizable yet user-friendly and non-obscure code generation.

### *A. Dynamic systems modeling*

As already mentioned, the user should be able to define the system using the SDL-RT language. This language is powerful enough and one is able to describe procedures, processes, blocks and systems using it. However, some of the parts of the target system have to be implemented manually; this is why we are talking about the "source code templates" generation, not the source code per se. At the moment, our software provides the user with the means of describing processes and procedures. It is planned to expand it with the means of process blocks, classes and systems definition.

In order to cut the development time and costs and to improve the user experience, we have chosen the wxShapeFramework [2] library as the diagram drawing engine. It is an open source library intended to be used with wxWidgets. While being simple enough to integrate with our project, it is a powerful, flexible and easy to customize tool that suits our needs perfectly. We have expanded it with our custom set of primitives according to the SDL-RT standard.

While the user draws the diagram, a corresponding graph is being built. This graph represents the structure of the diagram; it is an internal wxShapeFramework data structure. It is used to store the information about the diagram itself, i.e. the types of elements and relations between them; it also allows for such basic operations as file input/output, clipboard operations, undo/redo framework. Hence, it is of little interest for us.

At the same time, a complex system logics data structure is being built. It stores information about the system itself, as defined by the SDL-RT diagram. This information includes the definitions of the processes, procedures, variables, signals, gates etc. Every process, procedure, system etc. is described by the corresponding C++ class instance. Such classes store information about local data definitions, as well as the graphs that represent the algorithms being described. Every node of the graph stores its parameters, i.e. those that are specified by the SDL-RT standard.

While the system logics data structure is being built, it is being analyzed in real time; the errors and warnings are being reported to the user.

The diagrams are serialized in XML [3] files by the means of wxShapeFramework. Those are enough to store all the data needed to recreate the diagram, the locations and the parameters of all of the elements and their relations.

In order to provide our software with basic means of interoperability with existing and future third-party products, we are also working on a textual SDL-RT exporting routine according to the SDL-RT standard definition. Textual SDL-RT representation is actually an XML representation; its DTD can be found in the text of the SDL-RT standard.

### B. Customizable source code generation

When the system is defined and all the possible errors that prohibit code generation are eliminated, the user is able to launch the target code template generation routine. While designing this routine, we have faced several challenges; not all of our solutions are considered final yet and hence the information given here is a subject to change.

Our goal at this stage is to provide efficient, robust source code templates generation routine. Our specific requirement is that it should be flexible enough in order to both provide users with advanced means of controlling the code generation process and to allow for code generation for different target platforms and architectures.

While designing the code generation functionality, we kept in mind the fact that different real-time operating systems have different APIs and possibilities. However, in order to provide for implementation of the SDL-RT defined systems, a predefined set of functions should be supported by the target OS API. These functions include real-time process creation and destruction, fast signal-based IPC mechanism (including, at least, means of signal description, allocation, exchange and destruction), timers and semaphores. The debugging output functions are also needed. Most of the other actions described by SDL-RT are simple components of algorithms (flow control, loops, jumps etc) and thus are possible to implement using basic C89 [4] instructions.

Hence, it was decided to design an intermediate pseudo-language that should be able to describe both basic algorithms and basic IPC and synchronization functionality (signals, semaphores, timers). The textual SDL-RT representation is not quite suitable due to its structure (XML data). This pseudo-language source code is generated upon the user request when the system definition is finished and all the possible errors are eliminated.

This pseudo-language code may be stored in external files, but is not intended for user manipulation. As soon as this representation is built, the parser is run and the target C89 source code template is built.

In order to allow for source code generation for different target platforms (that obviously have different APIs), different rule sets should be applied to the generated source code. They describe the means of working with signals, timers and semaphores. Those rule sets are defined in external files and are available for user inspection and manipulation in order to provide further flexibility and extensibility of our software.

### C. Optional functionality

Apart from the system definition and source code templates generation, our software should also be able to implement other functionality. In order to allow for such extensibility, we have designed the plug-in architecture which is quite simple itself. The plug-in modules receive the definition of the system as generated on the step 1, perform their actions and report results to the user.

We are considering implementation of following plug-in modules: debugging, tracing, simulation, performance analysis (including bottlenecks detection), fine-tuning.

The overview of the workflow is given at the figure 1.

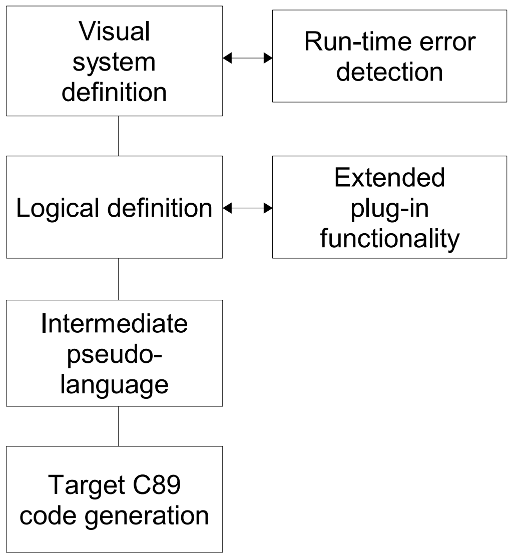

Visual system definition
Run-time error detection
Logical definition
Extended plug-in functionality
Intermediate pseudo-language
Target C89 code generation

Figure 1: Workflow overview

## III. Languages And Notations

There is a variety of protocol description's methods, languages, and algorithms. Most of them are governed by such well-known international organizations as ETSI, ITU, OMG, ISO etc. In order to achieve our goals, we had to choose the most appropriate, relevant, recognized ones.

We have chosen SDL-RT [1] as the core systems description language. The traditionally employed language, SDL [5], did not suit our exact needs due to the following drawbacks:

- Out-of-date data types and data description tools. ASN.1 [6] notation has a lot of advanced means of data types definition, but it is not quite convenient for definition of systems and for automated code generation.
- Out-of-date syntax. Formalization is not enforced enough (i.e. different standards descriptions that make use of SDL often have different conventions).
- No semaphores. It is especially inconvenient during real-time systems design.
- No pointers. Most of the embedded and real-time systems are implemented in C, which depends heavily on pointers.

SDL-RT is designed to eliminate the problems of classic SDL. Some of the original SDL features were eliminated as well, as they are almost never used in modern real-time systems design.

We have also chosen MSC [7] as a message flow visualization and debugging tool. It is also extended as a part of SDL-RT project.

## IV.Target Platforms And APIs

Actually, a target platform for the source code templates generated by our software is any OS (or RTOS) that both has C89-compliant compilers available and supports such common features as message exchange, semaphores and timers. A lot of different hardware platforms is currently supported by GNU/GCC [8] compilers. One of the specific requirements of our project is allowing users to tweak code generation rules, in order to make it possible to deploy the generated software for different target OS APIs. Our primary target API is ENEA LINX [9] library, which is intended to be used with Linux.

## V.Operating System, Tools And Libraries

We have chosen Linux as a host operating system. It has gained popularity during last few years and is widely recognized not only as an OS for servers or software developers, but also as a desktop OS. Being a free, open source operating system, Linux is a great yet low-cost software development environment. As a consequence, we have decided to use GNU/GCC g++ compiler.

Another problem we have faced was choosing the right GUI toolkit. The most popular ones are Qt [10], wxWidgets [11] and GTK+ [12]. All of them provide developers with similar possibilities, but we have chosen wxWidgets. It is free from license limitations (Qt was not licensed under LGPL at the time when the decision was made), it uses native GUI controls under target operating systems whenever possible, its build system is not as complicated as the one of Qt. wxWidgets also provides developers with auxiliary non-GUI classes, thus simplifying crossplatform development. Hence, our software will be portable and most likely available for Windows users.

## VI.Current Status And Future Prospect

At the current phase, an alpha version of the software was developed. It provides users with basic diagram drawing tools, error reporting and code generation tools.

A lot of work is being done to improve the code generation algorithms. We are planning to introduce new debugging features, real-time protocol simulation, performance estimations. We also plan to extend the list of supported target platforms to include the most popular and widely employed systems. Our long-term plans include implementing automated testing and test scenarios generation and detailed automated analysis of systems being designed. One of the possible long-term development plans is improving the project even further and releasing it as a commercial product.

## VII.Résumé

The proposed solution of the efficient dynamically changing systems design problem consists of the following:

- Correct decomposition of the problem,
- Thorough planning of data structures,
- Thorough modeling of algorithms,
- Using customizable source code generation routines,
- Using open development tools, libraries and solutions

## References

[1] SDL-RT. http://www.sdl-rt.org
[2] wxShapeFramework library. http://wxcode.sourceforge.net/components/shapeframework/
[3] XML. http://www.w3.org/XML/
[4] ANSI C. ANSI X3.159-1989 "Programming Language C."
[5] SDL. ITU-T Z.100
[6] ASN.1. ITU-T X.680
[7] MSC. ITU-T Z.120
[8] GNU/GCC. http://gcc.gnu.org/
[9] ENEA LINX. http://www.enea.com/Templates/Product____27016.aspx
[10] Qt. http://www.qtsoftware.com/
[11] wxWidgets. http://www.wxwidgets.org
[12] GTK+. http://www.gtk.org

# Application of UniTESK Technology for Functional Testing of Infrastructural Grid Software

Sergey Smolov

*ISP RAS*

ssedai@ispras.ru

## Abstract

*In this article some questions of testing of infrastructural Grid software on the standard compliance are discussed. Nowadays Grid-systems are one of the first-priority areas in computer science. The primary task is an effective usage of their advantages that is inseparably connected with the problem of software portability in Grid-systems. One of the simplest solutions of such problem is a development of standards on bundled software. Particularly, the compliance to the standard occurs as one of basic requirements to a system that is why its accomplishment should be checked with a high validity. In this letter the development of test patterns for infrastructural Grid software by means of UniTESK technology is considered. The result of program packet Globus Toolkit 4.2 testing upon compliance with WSRF 1.2 basic standard is given.*

## 1. Introduction

The term "Grid" appeared at the beginning of 1990th in "The Grid: Blueprint for a new computing infrastructure" [1] collection under the editorship of Ian Foster as a metaphor of such an ease of access to computational resources as to power grid. According to Ian Foster, Grid-system (hereinafter referred to as GS) is a system, that:

a) coordinates the usage of resources in the absence of a centralized management;

b) supports standard, open and universal protocols and interfaces;

c) non-trivially supplies with a high-quality services.

Following by this definition, GS is a universal infrastructure of processing and storing distributed data, and of different services, called Grid-services, that are functioning in it. Different GS must support standard protocols and interfaces, despite of possible differences in architecture or specialties of realization. The purpose of the Grid standardization is to guarantee a portability of applications between different Grids, including systems that are built upon different infrastructural program packages.

### 1.1 Grid standardization

Realizations of GS had been appearing since 1995, when infrastructural program package Globus Toolkit appeared; nowadays it is de facto a standard of GS. It was made by Globus Alliance – the major international consortium in the area of Grid. By 1997 a European project of creating a program package for GS had begun, and it has brought to the infrastructural UNICORE software. By 2004 under the aegis of EGEE (Enabling Grids for E-sciencE) project, the gLite package had been released. In the connection with the great number of incompatible realizations of infrastructural Grid software, the necessity of unification and standardization became actual and active work on Grid standards creation started. Three groups of standards of the infrastructural Grid software are existing nowadays: the WSRF (Web Services Resource Framework), the OGSA (Open Grid Services Architecture) and the WS-Management.

### 1.2 Questions of Grid realizations testing

One of the specialties of the application domain is an existing of the incompatible, generally speaking, standards and some independent realizations. It is clear, that testing of the realization upon the standard compliance is a very actual problem.

It is worth to notice, that in the application domain the specificity of interaction between user applications and GS means occurs – in this case they are Grid- and Web-services and remote procedure calls.

The most widespread approaches to the GS testing are:

a) unit testing – a testing of different software modules. Particularly, Globus Toolkit developers use

JUnit for their realization checking.

b) integration testing – a testing of applications execution upon the infrastructural Grid software assembly. Typical examples of integration test scenarios are data bulk transfers and routine calculation implementations.

Both approaches are turned to realization errors detection. But they have a considerable defect in the context of compliance testing: there is no connection between tests and standards requirements. That is, it is impossible to draw a conclusion about compliance or mismatching to the standard by test results.

### 1.3 Technology UniTESK of the automated testing

Since 1994 the UniTESK technology of the automated testing has been developed. It was successfully used for testing of the different classes of program systems – program interfaces, telecommunication protocols and hardware. An access to the infrastructural Grid software is realized by different mechanisms of remote procedure calls and official protocols that are very close to the domain of applicability of the UniTESK, that's why this technology was chosen as a technology platform for tests development.

The test development with the application of the UniTESK technology accomplishes in the next seven stages:

1) requirement analysis and its formalization, formal specifications building;

2) requirements to the quality of testing formulation;

3) test scenarios, that are realizing such coverage, development;

4) test scenarios binding to the concrete target system by mediator development;

5) tests translation and compilation;

6) tests debugging and execution;

7) testing results analysis.

The important advantage of the UniTESK technology is an estimation of quality of testing possibility by calculating the requirements coverage. That is also the essential distinction from overwhelming majority of test patterns for GS. The calculation of requirements coverage and the constructing of oracles (special components that are checking the compliance of the target system to the specification) are automated.

Thus, the UniTESK technology allows driving widespread testing of a big class of program systems and, particularly, to develop test patterns for compliance problem solution.

### 1.4 OGSA and WSRF standards. Applicable domain of problem restriction.

The OGSA standard describes infrastructural middleware OGSI (Open Grid Services Infrastructure) between user applications and computational resources. It means that applications have no any possibility to interact with resources directly, but only by using infrastructural software. At the root of OGSI architecture the Grid-service concept lies that represents a mechanism of remote calls and was developed specially for Globus Toolkit 3.

At the same time with OGSA standard development, in 2004 the OASIS consortium suggested the WSRF standard. It also describes some infrastructural software, but based on not Grid- but Web-services. Correspondingly, the application domain of this research is exactly infrastructural software that is based on Web-services, because on this concept the most widespread infrastructural Grid software realizations are based.

In this research a possibility of UniTESK technology application for the functional testing of Grid software analyses, including the testing of standard compliance. Particularly, the next problems are considered:

1) representation of requirements to the services of infrastructural Gird software in formal UniTESK specifications;

2) development of mediators for impacts upon infrastructural Grid software;

3) development of testing scenarios for infrastructural Grid software.

## 2. Requirements to the realizations of infrastructural Grid software formalization

### 2.1 Regulating documents and requirements

**to the realizations of infrastructural Grid software**

As it is mentioned above, there are two standards that are used in GS development nowadays: OGSA and WSRF. The OGSA standard is a description standard, it does not contain any functional requirements, existing requirements are uncertainly expressed, descriptions of message formats and remote accessing protocols are not given. Therefore, the OGSA standard does not suit to be the base for the formal specification and based on it test pattern development.

The WSRF standard for standard formalization suites more. It contains 5 specifications:

1) WS-BaseFaults – determines format of error messages and the mechanism of their processing;

2) WS-Resource – determines WS-Resource concept itself, formats of messages and the semantics of management services;

3) WS-ResourceLifetime – determines mechanisms of destroying the WS-Resource;

4) WS-ResourceProperties – determines, in what way a WS-Resource is connected with an interface, that describes Web-service, and also represents the mechanisms of getting, changing and deleting properties of WS-Resource;

5) WS-ServiceGroup – determines an interface to the set of heterogeneous Web-services.

The Resource is a logical entity that has the following characteristics: identifiability, lifetime and set of zero or more properties, which are expressible in XML Infoset. The WS-Resource is a composition of the Resource and the Web-service, by using its methods or fields an access to the Resource is accomplished. The WSRF standard is very convenient to analyse because of its structuredness. For example, the bigger part of functional requirements are supplied with RFC 2119 keywords – MUST, SHOULD, MAY. Moreover, most requirements are supplied with blocks of descriptions and examples that have been written on pseudocode that looks like WSDL 2.0 Web-services description language.

Certain parts of standard have different levels of obligatory in RFC 2119 gradation. That is, a WS-Resource must realize requirements of WS-Resource and WS-ResourceProperties specifications, also it should realize requirements of WS-Base-Faults specification and it may satisfy requirements of WS-ResourceLifetime. Consequently, under the test pattern development, the first two specifications should be taken into account first of all. Such approach considerably simplifies the test pattern structure and reduces it, but retains tests correctness as solution of the problem of compliance.

There are following message exchanges in WS-ResourceProperties specification:

1) GetResourcePropertyDocument – getting all the properties of the WS-Resource;

2) GetResourceProperty – getting the certain property of the WS-Resource;

3) GetMultipleResourceProperties – getting multiple properties of the WS-Resource;

4) QueryResourceProperties – determining the structure of WS-Resource properties and querying the requests upon them;

5) PutResourcePropertyDocument – changing of the “old” Resource Property Document (a set of all properties of the WS-Resource) by the “new” one;

6) SetResourceProperties – changing some properties of the WS-Resource (i.e. a composition of the three following message exchanges);

7) InsertResourceProperties – adding new properties;

8) UpdateResourceProperties – changing values of the existing properties of the WS-Resource;

9) DeleteResourceProperties – deleting some properties of the WS-Resource.

In the course of the analysis of standard 325 functional requirements had been marked out, 29 – in WS-BaseFaults specification, 12 – in WS-Resource, 51 – in WS-ResourceLifetime, 159 – in Ws-ResourceProperties and 73 – in WS-ServiceGroup.

## 2.2 Formal specification development

Every message exchange in WSRF corresponds to pair <request - response>, where in the capacity of the response can be message with the returned value or error message. Thereby WSRF message exchanges can be modeled as function calls with the returned values, error messages can be modeled as exceptions.

Due to in the concerned method of the requirements to the infrastructural Grid software formalization, the Web-service is modeled as an object of some class, and message exchanges between client and realization are represented as calls of this class methods.

Altogether a half of all WSRF requirements was formalized (nearly 160 requirements) and about 60% requirements from WS-Resource, WS-

ResourceLifetime and WS-ResourceProperties specifications. The whole set of requirements can be separated into two basic groups: syntactical and functional requirements. Syntactical requirements impose constraints on a structure of messages and relationships between the fields of one message. Functional requirements correspond as restrictions upon the functionality of requests processing and connections between a content of request and a content of response. It is recommended to check syntactical requirements in mediators at the stage of message parsing.

### 2.3 Mediator development

The main function of mediator, as a component of a test pattern, is an establishment of correspondence between a model object (object of specification class) and a target system. In case of infrastructural Grid software testing, there is no possibility to have an access to fields and methods of system, that is why it could be accomplished only by sending appropriate requests and receiving and parsing responses. In concerned method mediator transforms parameters of a specification method into SOAP/HTTP message and sends it to the target system on established TCP connection. Received responses are checked by the mediator, the correspondence to syntactical requirements and parsed by mediator too. Then mediator forms the returned value of the specification function.

The peculiarity of the mediator development is that existing realizations (particularly, Globus Toolkit 4.2) do not satisfy to syntactical requirements of the standard. For the testing implementation the adoption of mediators by standard violations was needed. Also by analyzing the source code of infrastructural Grid software Globus Toolkit 4.2 and taking some experiments with realization was established that realization does not support the following message exchanges – PutResourceProperties and SetResourceProperties.

### 2.4 Test scenarios development

Under test scenario for testing of the infrastructural Grid software of compliance to the WSRF standard development, in the capacity of an automate state identifier it is recommended to use the cardinality of Resource Property Document of the WS-Resource, i.e. an amount of WS-Resource properties. On the one hand, such feature considerably reduces the complexity of supposed state graph, and, appropriately, a time of tests execution. On the other hand, under such definition of state of the automate the determinacy of graph retains that is important for a correct operation of the UniTESK iterator.

Transitions between automate states are accomplished by offering stimuli into the target system. The role of stimuli in the case of development of the test pattern for infrastructural Grid software play mediator methods calls. As soon as synchronous model of the target system is used, for every specification method it is possible to create a separate scenario method, which will go over the parameters of method and will call (implicitly) mediator for testing impact and an oracle for checking the returned value on correctness.

## 3. Experience of practical testing of the infrastructural Grid software

The method, which was represented above, was used under development of the test pattern for checking the compliance of the infrastructural Grid software realizations to the WSRF standard.

### 3.1 Target system review

The object under testing in this research is a program packet Globus Toolkit 4.2. Globus Toolkit 4.2 uses protocols of standard Web-services and mechanisms of services description, detection, control, authentication and authorization. This program packet includes components that can be used for constructing containers. In these containers Web-services, which are written on Java, C and Python can be placed.

In accordance with Globus Alliance, the Java WS Core component of Globus Toolkit 4.2 (it supports the Web-services development and execution of Java applications) realizes requirements of WSRF standard. In this research the problem of compliance was decided exactly for this component.

### 3.2 Implementation testing

Test scenarios that were developed by using JavaTESK instrument include scenarios for eight methods: GetResourcePropertyDocument, GetResourceProperty, GetMultipleResourceProperties

Insert-, Update- and DeleteResourceProperties, and also ImmediateDestroy and SheduledDestroy.

It is worth explain, why these eight message exchanges (i.e. specification methods) were chosen. At the time of test pattern development it was supposed, that by test pattern using all services of the container of the Java WS Core component will be tested. By default, there is 34 Web-services in container, and only 23 of them support message exchanges, that are mentioned above and are contained in WS-ResourceProperties and WS-ResourceLifetime specifications of WSRF standard. Correspondingly, these 8 message exchanges is both simple enough to specification methods development (the QueryResourceProperties message exchange is not so simple for testing) and allow to run testing of the compliance to the WSRF 1.2 standard.

### 3.3 Testing results

In this research the realization of infrastructural Grid software Globus Toolkit was tested by the facilities of the UniTESK (JavaTESK) technology. In the capacity of testing results reports about requirements coverage, which were generated by the instrument, acted. Nearly 60% of system functionality was covered, wherein this test pattern does not yield to tests that are used by Globus Toolkit developers. They measures code coverage by tests for quality of testing determination with the JUnit and Clover instruments. These instruments allowed developers to determine that their unit-tests had covered 60% of the functionality of system too (i.e. Java WS Core component). However, as it was mentioned above, these tests cannot report about the compliance of Realization Globus Toolkit 4.2 to the some standard. Thus, existing and developed tests not only complement each other, but allow considering the realization from the different points of view.

Under the realization testing some semantic discrepancies to the WSRF 1.2 standard (in InsertResourceProperties and UpdateResourceProperties message exchanges) were revealed. Particularly, these methods allow to add and change values of properties of the WS-Resource with different identifiers (which are called QNames) that is forbidden by the standard requirements.

## 4. Existing methods and approaching of Grid infrastructural software testing

In 2006 ETSI (European Telecommunications Standards Institute) organized an expert group for development a test pattern for Grid compliance testing [2]. Method that is being developed in ETSI is based on the development on large amount of test cases on TTCN-3 programming language. The prototype of such test pattern on TTCN-3 language is represented in [3]. Test pattern is not connected with any standard of infrastructural Grid software. Instead of standard requirements checking this test pattern checks applicability of Grid in typical use cases – statement of the computational task in query, task execution on one of the computing nodes, result delivery.

In [4] authors propose an approach to the Grid testing that is close to the approach presented in this article. Authors offer to use a formalism of abstract state machines (ASM) and automatically generate test sequences from automate bypass. Questions of Grid standards analysis and requirements formalization and formal model building are not considering.

## 5. Conclusion

In this research the problem of testing of the compliance of the infrastructural Grid software realization Globus Toolkit 4.2 to the standard WSRF 1.2 was being solved. This standard was analyzed and a catalogue of its requirements was created. Interfaces and the structure of Java WS Core component of the Globus Toolkit 4.2 were explored also. On basis of these data the test pattern for this program packet was developed by using UniTESK (JavaTESK) technology and testing was carried out. The testing has showed that the realization Globus Toolkit 4.2 complies with the WSRF standard and has revealed a lack of some unnecessary requirements accomplishment, both functional and syntactical. Also testing has showed that the UniTESK technology is applicable for testing the infrastructural Grid software, particularly, there are the following peculiarities of its application:

1) message exchanges with Web-services are essentially modeled by specification functions;

2) for test pattern development a class library for automatization of building different (and “incorrect” also) messages of Web-services is necessary.

In the capacity of directions for the future research we consider a development of test pattern, that is checking a specific requirements to the services of the infrastructural Grid software, like a service of bulk data transferring, service of creation and management of computational resources and so on.

## 6. References

[1] Ian Foster *The Grid: Blueprint for a New Computing Infrastructure.* — Morgan Kaufmann Publishers. — ISBN 1-55860-475-8

[2] S. Schulze. Achieving Grid Interoperability: The ETSI Approach. *The 20th Open Grid Forum - OGF20/EGEE 2nd User Forum.* Manchester, UK. May 7 - 11, 2007

[3] T.Rings, H.Neukirchen, J.Grabowski. Testing Grid ApplicationWorkflows Using TTCN-3. First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008.

[4] Lamch, D.; Wyrzykowski, R. Specification, Analysis and Testing of Grid Environments Using Abstract State Machines. *International Symposium on Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006.* 13-17 Sept. 2006 Pages:116 - 120

# Test data generation for LRU cache-memory testing

Evgeni Kornikhin
Moscow State University, Russia
Email: kornevgen@gmail.com

***Abstract*—System functional testing of microprocessors deals with many assembly programs of given behavior. The paper proposes new constraint-based algorithm of initial cache-memory contents generation for given behavior of assembly program (with cache misses and hits). Although algorithm works for any types of cache-memory, the paper describes algorithm in detail for basis types of cache-memory only: fully associative cache and direct mapped cache.**

## I. Introduction

System functional testing of microprocessors uses many assembly programs (*test programs*). Such programs are loaded to the memory, executed, execution process is logged and analyzed. But modern processors testing requires a lot of test programs. Technical way of test program generation was proposed in [1]. This way based on the microprocessor's model. Its first stage is systematic generation abstract test programs (*test templates*). This abstract form doesn't contain initial state of microprocessor but contain sequence of instructions with arguments (registers) and with *test situations* (behavior of this instruction; these can be overflow, cache hits, cache misses). The second stage is generation of initial microprocessor state for given test template. This stage is test data generation. Technical way from [1] is useful for aimed testing when aim is expressed by instruction sequence with specific behavior. Initial microprocessor state includes initial values of registers and initial contents of cache-memory. Based on this state the third, final, stage is generation the sequence of instructions to reach initial microprocessor state. These sequence of instructions with test template get ready assembly program. This paper devoted to the second stage, i.e. initial state generation.

Known researches about test data generation problem contain the following methods of its solving:

1) combinatorial methods;
2) ATPG-based methods;
3) constraint-based methods.

Combinatorial methods are useful for simple test templates (each variable has explicit directive of its domain, each value in domain is possess) [2]. ATPG-based methods are useful for structural but not functional testing [3]. Constraint-based methods are the most promising methods. Test template is translated to the set of constraints (predicates) with variables which represented test data. Then special solver generates values for variables to satisfy all constraints. This paper contains constraint-based method also. IBM uses constraint-based method in Genesys-Pro [4]. But it works inefficiently on test templates from [1]. Authors of another constraint-based methods restrict on registers only and don't consider cache-memory.

## II. Test templates description

Test template defines properties of future test program. Test template contains sequence of instructions. Each element of this sequence has instruction name, arguments (registers, addresses, values) and test situation (relation between values of arguments and microprocessor state before execution of instruction). Example of test template description for model instruction set:

```
REGISTER reg1 : 32;
REGISTER reg2 : 32;
ADD reg1, reg2, reg2
LOAD reg1, reg2 @ l1Miss, l2Hit
SUB reg2, reg1, reg2
```

This template has 3 instructions – ADD, LOAD and SUB. Template begins from variable definitions (it has name of variable and its bit length). Test situation is specified after "@": test situation of the second instruction is "l1Miss, l2Hit": "l1Miss" means cache miss in first-level cache and "l2Hit" means cache hit in second-level cache.

Model instruction set contains only 2 memory operation:

- "LOAD reg, address" loads value from memory by physical address "address" to the register "reg";
- "STORE reg, address" stores value from register "reg" to the memory by physical address "address".

Test data generation is generation of initial values of registers and initial contents of cache-memory. This problem has been solved for common microprocessor cache-memory. The following consists of test data generation for 2 basis cache-memory organizations: fully associative cache with LRU and direct mapped cache. Common cache includes aspects from both cache-memory organizations. The rest of paper deals with one-level cache-memory although proposed method can be applied to cache memory with more than one level.

## III. Test data generation for fully associative cache

*Fully N-associative cache* consists of N cells (N means *cache associativity*). Each cache cell may store data from any memory cell. All cache cells correspond to the different memory cells. Access to memory starts from access to cache. Search data in cache performs for each cache cells in parallel. *Cache hit* means existence data in cache. *Cache miss* means absence of data in cache. In case of cache miss one cache

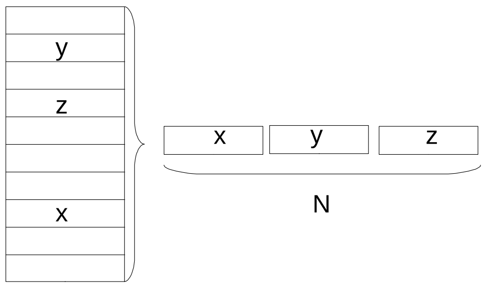


Fig. 1. Fully N-associative cache

cell must be replaced on data from required address by specific *replacement strategy*. This paper uses LRU replacement strategy (Least Recently Used). According to LRU the least recently used cache cell will be evicted. At the following phrase ”evicted address $x$” means evicted data by address $x$.

Proposed algorithm based on the following properties of evicted addresses:

1) any evicted address was inserted by instruction from test template with cache miss or was in the initial contents of cache;
2) between replacing and the last access to the same address (cache hit or cache miss) there are accesses to the whole cache without address itself.

Proposed algorithm generates constraints on the following variables:

1) $\alpha_1, \alpha_2, ..., \alpha_N$ – initial contents of cache (its count equals to cache associativity);
2) hits-addresses (addresses of instructions from test templates with cache hit test situation);
3) misses-addresses (addresses of instructions from test templates with cache miss test situation);
4) evicted addresses (evicted addresses of instructions from test templates with cache miss test situation);
5) $L_0, L_1, ...$ – cache states

Each instruction from test template with cache hit gives 1 new variable, and each instruction with cache miss gives 3 new variable (1 for miss address, 1 for evicted address, and 1 for cache state). Proposed algorithm generates constraints for each instruction from test template by the following ($N$ means cache associativity):

1) ”initial constraints” are generated one time for any test template: $L_0 = \{\alpha_1, \alpha_2, ..., \alpha_N\}$, $|L_0| = N$ (other words, numbers $\alpha_1, \alpha_2, ..., \alpha_N$ are different);
2) ”hit-constraints” are generated for each instruction from test template with cache hit: $x \in L$, when $x$ means address from instruction, $L$ means a current cache state-variable;
3) ”miss-constraints” are generated for each instruction from test template with cache miss ($x$ means evicting address, $y$ means evicted address, $L$ means a current cache state-variable): $y \in L, x \notin L, L' = L \cup \{x\} \setminus \{y\}, lru(y)$, $L'$ became a current cache state-variable for the next instruction.

Constraint $lru(y)$ defines $y$ as the least recently used address.

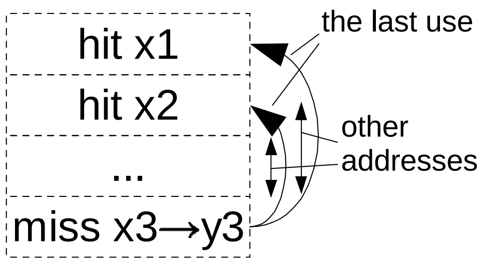


Fig. 2. LRU

Constraint $lru(y)$ is disjunction of constraints corresponded to cases of the last access to the $y$ before its eviction. Each its clause is conjunction of the following constraints ($x$ means the address-variable from the last access to the $y$):

1) $x = y$
2) $L \setminus \{y\} = \{x_1, x_2, ..., x_n\}$, where $x_1, x_2, ..., x_n$ are all addresses accessed between accesses to $x$ and $y$ (hits and misses).

The last access to the $y$ can correspond to the previous instruction of test template or to the cell from initial cache state.

Consider an example of test template and its test data generation for 3-associative cache.

LOAD x, y @ Hit
STORE u, z @ Miss
LOAD z, y @ Hit

Define unique names for variables in test template (each new variable shouldn't change its value). LOAD gives new version for its first argument. STORE doesn't generate new version of variables. Define new variable $z'_0$ for evicted address from the second instruction (this variable won't be included to the solution):

LOAD $x_1, y_0$ @ Hit
STORE $u_0, z_0$ @ Miss $\rightarrow z'_0$
LOAD $z_1, y_0$ @ Hit

Define variables for initial contents of cache: $\{\alpha, \beta, \gamma\}$ (its count equals to cache associativity).

So the task is looking for values of $x_0, y_0, z_0, u_0, \alpha, \beta, \gamma$ according to test template. This task has more than 1 solutions. But any solution is enough.

The first constraints describe cache hits and misses as belong to the current state of cache:

$y_0 \in \{\alpha, \beta, \gamma\}$,
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$z'_0 \in \{\alpha, \beta, \gamma\}$,
$y_0 \in \{\alpha, \beta, \gamma\} \setminus \{z'_0\} \cup \{z_0\}$,
$\alpha, \beta, \gamma$ – different

Define constraint $lru(z'_0)$. Candidates of the last access to the this address are $y_0, \gamma, \beta, \alpha$. The first and the second candidates aren't suitable because constraint $L \setminus \{z'_0\} = X$ is false because of different compared sets capacity. Remainder candidates give the following disjunction:

$z'_0 = \beta \wedge \{\alpha, \beta, \gamma\} \setminus \{z'_0\} = \{\gamma, y_0\}$
$\vee$
$z'_0 = \alpha \wedge \{\alpha, \beta, \gamma\} \setminus \{z'_0\} = \{\beta, \gamma, y_0\}$
Simplify it:
$z'_0 = \beta \wedge \{\alpha, \gamma\} = \{\gamma, y_0\}$
$\vee$
$z'_0 = \alpha \wedge \{\beta, \gamma\} = \{\beta, \gamma, y_0\}$
Further simplify:
$z'_0 = \beta \wedge y_0 = \alpha$
$\vee$
$z'_0 = \alpha \wedge y_0 \in \{\beta, \gamma\}$

Consider the first clause with the rest of constraints (variable $z'_0$ isn't needed in solution):
$y_0 = \alpha$
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$\alpha, \beta, \gamma$ – different

Note that $x_0$ and $u_0$ don't take part in constraints. So their values may be arbitrary.

Lets bit length of addresses is 8. So domain of all variable-addresses is from 0 to 255. Satisfying constraints variables can get the following values (these values are not unique):
$\alpha = y_0 = x_0 = u_0 = 0$
$\beta = 1$
$\gamma = 2$
$z_0 = 3$

Verify test template execution with computed initial cache state and register values:
initial cache state is [2, 1, 0]
LOAD x, 0 - Hit, because $0 \in \{2, 1, 0\}$; according to LRU the next cache state is [0, 2, 1]
STORE 0, 3 - Miss, because $3 \notin \{0, 2, 1\}$; according to LRU 3 goes to cache, 1 is evicted from cache, the next cache state is [3, 0, 2]
LOAD z, 0 - Hit, because $0 \in \{3, 0, 2\}$
All instructions from test template were executed according to given test situations.

## IV. Test data generation for direct mapped cache

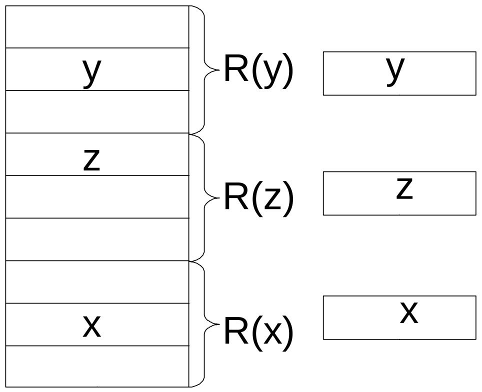


Fig. 3. Direct mapped cache

Whole memory is divided into non-intersecting areas (*regions*). Direct mapped cache consists of 1 cell for each region. Each cache cell may store data only from its region. Access to memory starts from access to cache. *Cache hit* means successful match cached address with required address in its region. *Cache miss* means unsuccessful match cached address with required address in its region. In this case data from cache replaced by data from memory by required address.

Proposed algorithm generates constraints on the following variables:

1) $\alpha_1, \alpha_2, \alpha_3, ...$ are addresses of the initial cache state (their count is regions' count);
2) hits-addresses (addresses of instructions from test templates with cache hit test situation);
3) misses-addresses (addresses of instructions from test templates with cache miss test situation);
4) evicted addresses (evicted addresses of instructions from test templates with cache miss test situation);
5) $L_0, L_1, ...$ – cache states

Define function $R(y)$ which for address $y$ returns a set of all cells from the same region as region of $y$. $R$ satisfies the following properties:
$\forall x \ (x \in R(x))$
$\forall x \ \forall y \ (x = y \rightarrow R(x) = R(y))$
$\forall x \ \forall y \ (R(x) = R(y) \leftrightarrow x \in R(y))$
$\forall x \ \forall y \ (R(x) = R(y) \leftrightarrow y \in R(x))$
$\forall x \ \forall y \ (x \notin R(y) \rightarrow x \neq y)$

Proposed algorithm generates constraints for each instruction by the following way ($N$ means number of regions):

1) "initial constraints" are generated one time for each template : $|\{\alpha_1, \alpha_2, ..., \alpha_N\}| = N$ (other words, numbers $\alpha_1, \alpha_2, ..., \alpha_N$ are different), $|\{R(\alpha_1), R(\alpha_2), ..., R(\alpha_N)\}| = N$ (other words, all sets $R(\alpha_1), R(\alpha_2), ..., R(\alpha_N)$ are different);
2) "hits-constraints" are generated for each instruction with cache hit: $x \in L$, where $x$ means address from instruction, $L$ means a current variable-state of cache memory;
3) "miss-constraints" are generated for each instruction with cache miss ($x$ means evicting address, $y$ means evicted address, $L$ means a current variable-state of cache): $y \in L, x \notin L, L' = L \cup \{x\} \setminus \{y\}, R(y) = R(x)$, $L'$ became the current variable-cache state for the next instruction.

Constraints for direct mapped cache differ from constraints for fully associative cache by evicted address constraints only.

Consider test data generation for the already known test template. Lets memory divided into 3 regions depended on remainder from division address to 3 (i.e. $R(x) = R(y) \Leftrightarrow 3|(x - y)$ ).
LOAD x, y @ Hit
STORE u, z @ Miss
LOAD z, y @ Hit

Define unique names for variables in test template (each new variable shouldn't change its value). LOAD gives new version for its first argument. STORE doesn't generate new version of variables. Define new variable $z'_0$ for evicted address from the second instruction (this variable won't be included to the solution):

LOAD $x_1, y_0$ @ Hit
STORE $u_0, z_0$ @ Miss $\rightarrow z'_0$
LOAD $z_1, y_0$ @ Hit

Define variables of initial cache state: $\{\alpha, \beta, \gamma\}$ (one for each region).

So the task is looking for values of $x_0, y_0, z_0, u_0, \alpha, \beta, \gamma$ according to test template. This task has more than 1 solutions. But any solution is enough.

The first constraints describe cache hits and misses as belong to the current state of cache:

$y_0 \in \{\alpha, \beta, \gamma\}$,
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$z'_0 \in \{\alpha, \beta, \gamma\}$,
$y_0 \in \{\alpha, \beta, \gamma\} \setminus \{z'_0\} \cup \{z_0\}$,
$R(z_0) = R(z'_0)$,
$\alpha, \beta, \gamma$ – different
$R(\alpha), R(\beta), R(\gamma)$ – different

Simplify this constraints set:

$z'_0 \in \{\alpha, \beta, \gamma\}$,
$y_0 \in \{\alpha, \beta, \gamma\} \setminus \{z'_0\}$,
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$3|(z_0 - z'_0)$,
$\alpha, \beta, \gamma$ – different
$R(\alpha), R(\beta), R(\gamma)$ – different

Note that $x_0$ and $u_0$ don't take part in constraints. So their values may be arbitrary.

Lets bit length of addresses is 8. So domain of all variable-addresses is from 0 to 255. Satisfying constraints variables can get the following values (these values are not unique):

$\alpha = x_0 = u_0 = 0$
$\beta = y_0 = 1$
$\gamma = 2$
$z_0 = 3$

Verify test template execution with generated initial cache state and register values:

initial cache state is $L = [(R = 0) \mapsto 0, (R = 1) \mapsto 1, (R = 2) \mapsto 2]$

LOAD x, 1 - Hit, because $R(1) = L[R = (1 \ mod \ 3)]$

STORE 0, 3 - Miss, because $R(3) \neq L[R = (3 \ mod \ 3)]$, 1 is evicted from cache, the next state of cache is $L = [(R = 0) \mapsto 0, (R = 1) \mapsto 3, (R = 2) \mapsto 2]$

LOAD z, 0 - Hit, because $R(0) = L[R = (0 \ mod \ 3)]$

All instructions from test template were executed according to given test situations.

## V. Test data generation for common cache

This section consists of illustration only the constraints for common cache.

Define function $R(x)$ as the same as for direct mapped cache.

Consider known test template for memory consisted of 3 regions ($R(x) = R(y) \leftrightarrow 3|(x - y)$) of 2-associative cache:

LOAD x, y @ Hit
STORE u, z @ Miss
LOAD z, y @ Hit

Define unique variables (and $z'_0$ for evicted address):

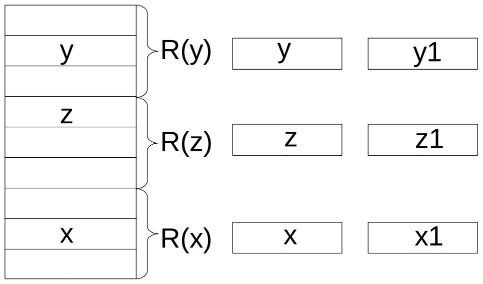


Fig. 4. Common cache

LOAD $x_1, y_0$ @ Hit
STORE $u_0, z_0$ @ Miss $\rightarrow z'_0$
LOAD $z_1, y_0$ @ Hit

Define variables for initial cache state: $\alpha_1, \alpha_2$ for the first region, $\beta_1, \beta_2$ for the second region, $\gamma_1, \gamma_2$ for the third region. Constraints set is the following:

$y_0 \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\}$,
$z'_0 \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\}$,
$z_0 \notin \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\} \cap R(z'_0)$,
$y_0 \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\} \cup \{z_0\} \setminus \{z'_0\}$,
$R(z_0) = R(z'_0)$,
$\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ – different,
$R(\alpha_1) = R(\alpha_2)$,
$R(\beta_1) = R(\beta_2)$,
$R(\gamma_1) = R(\gamma_2)$,
$R(\alpha_1), R(\beta_1), R(\gamma_1)$ – different

From disjunction for $lru(z'_0)$ (one clause is enough):

$z'_0 = \gamma_2 \wedge (\{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\} \setminus \{z'_0\}) \cap R(z'_0) = \{y_0\} \cap R(z'_0)$
$\vee$
...

Simplify:

$y_0 \in \{\alpha_1, ..., \gamma_2\}$,
$z'_0 \in \{\alpha_1, ..., \gamma_2\}$,
$z_0 \notin \{\alpha_1, ..., \gamma_2\} \cap R(z'_0)$,
$y_0 \in \{\alpha_1, ..., \gamma_2, z_0\} \setminus \{z'_0\}$,
$R(z_0) = R(z'_0)$,
$z'_0 = \gamma_2$,
$\{\gamma_1\} = \{y_0\} \cap R(\gamma_2)$
$\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ – different,
$R(\alpha_1) = R(\alpha_2)$,
$R(\beta_1) = R(\beta_2)$,
$R(\gamma_1) = R(\gamma_2)$,
$R(\alpha_1), R(\beta_1), R(\gamma_1)$ – different

Further simplify:

$z'_0 = \gamma_2$,
$y_0 = \gamma_1$,
$z_0 \notin \{\gamma_1, \gamma_2\}$,
$R(z_0) = R(\gamma_2)$,
$\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ – different,
$R(\alpha_1) = R(\alpha_2)$,
$R(\beta_1) = R(\beta_2)$,
$R(\gamma_1) = R(\gamma_2)$,
$R(\alpha_1), R(\beta_1), R(\gamma_1)$ – different

Lets bit length of addresses is 8. So domain of all variable-addresses is from 0 to 255. Satisfying constraints variables can get the following values (these values are not unique):

$\alpha_1 = 0, \alpha_2 = 3,$

$\beta_1 = 1, \beta_2 = 4,$

$\gamma_1 = 2, \gamma_2 = 5,$

$x_0 = 0, y_0 = 2, z_0 = 7, u_0 = 0$ .

Special algorithms can be used for solving constraints set. These algorithms can take into account the following aspects:

- constraints can be solved symbolically;
- all sets of addresses are finite and subset of all initial cache state addresses union with evicting addresses.

## VI. Conclusion

The paper devoted to the test data generation problem. Test data contains initial contents of cache-memory. The paper has proposed the constraint-based algorithm. Constraints consists of finite sets variables and sets operations. Test data generation for fully associative cache and direct mapped cache has been considered in details. Proposed algorithm is used in projects of testing MIPS-compatible microprocessors. ECLiPSe is used as constraint solver.

## References

[1] A.S. Kamkin, *Test program generation for microprocessors* // Proceedings of ISP RAS. Vol. 14(2). P.23-64. 2008.

[2] K. Takayama, F. Fallah, *A new functional test program generation methodology* // Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors. P.7681. 2001.

[3] F. Ferrandi, D. Sciuto, M. Beardo, F. Bruschi, *An approach to functional testing of vliw architectures* // Proceedings of the IEEE International High-Level Validation and Test Workshop (HLDVT00). P.2933. 2000.

[4] Y. Lichtenstein, M. Rimon, M. Vinov, M. Behm, J. Ludden, *Industrial experience with test generation languages for processor verification* // Proceedings of the 41st Design Automation Conference (DAC04). 2004.

# Model-based Technology of Automated Performance Testing

Prof. Dr. B. Pozin
*ZAO "EC-leasing"*
*bpozin@ec-leasing.ru*

R. Giniyatullin
*ZAO "EC-leasing"*
*renat@ec-leasing.ru*

Dr. I. Galakhov
*ZAO "EC-leasing"*
*igalakhov@ec-leasing.ru*

D. Vostrikov
*ZAO "EC-leasing"*
*dvostrikov@ec-leasing.ru*

## Abstract

*The Technology of the scaled-down experiment for evaluation of the performance characteristics of wide class of systems for business-process automation has been developed. The Technology objectives are to provide adequacy of the results of particular experiment and its components: problem definition, source data and analysis of results of the experiment. The Technology is based on the set of metamodels that define main components of the experiments. Technology provides technique for adaptation of metamodels to the parameters of any particular automated performance testing experiment.*

## 1. Introduction

The major works in the area of Model-based testing (MBT) are mainly aimed at the use of various statistics programs to generate tests data or to estimate tests results at automated functional testing. Detailed analysis of these works as well as creation and usage methods at MBT is given in the papers of Whittaker J.A. [5], Petrenko A. [6] and others. As for functional testing, the use of models, reflecting tested programs statistics properties (structure, data flow) is highly effective. Model dimension can reach up to hundreds and even thousands of vertexes.

In this work the model-based approach is treated in the connection with information system (IS) performance testing. Within its framework application software testing in IS is carried out in functioning environment, that is, in the environment, including DBMS, data base, system software, hardware.

Few works are devoted to model-based performance testing methods. Model-based methods are most frequently applied in case a large number of virtual clients generation is involved in the course of web-systems and servers system testing in order to define their workload capacity [7]. Service state change structure, represented by web-system taking into account service realization probabilities and their sequence, is being modeled. There are also papers on model creation for networks analysis at their performance testing. [8]. Such a universal model, for example, allows to unite different testing tools into unified technology on the basis of several types, so called Universal Probe (UP), that define architecture, system element interaction policy and resources constraints.

However, as it is seen from practical testing of critical IS, the above said models do not cover performance testing planning and evaluating adequacy of results.

IS performance testing in the operational process is mainly aimed at **evaluation of operational characteristics of information system** (IS, in the composition of which the application software) not at application software functioning accuracy (as it is assumed, application software functional testing has been performed). More often than not, these characteristics are called "performance characteristics", or non-functional requirements, that is, productivity, responsiveness at functional tasks solution and others.

The main complexity at performance testing is source data and IS performance test results adequacy provision according to the customer understanding. The problem is, that the customer is not always able to define objectives of performance testing and to comprehend in advance what result can and must be drawn at the end of testing before it takes place. Thus, planning and execution of performance experiment can last up to two or three months. Inaccurate problem definition can cause repeated performance experiments, the cost of each being rather high.

For performance testing quality improvement and cost reduction it is necessary to introduce automated performance testing complex technology, that:

- is comprehensible both for the customer and for the contractor,
- provides documenting opportunities
  - of all testing scenarios;
  - of structure and workload composition suppositions;
  - of the tested system composition and structure as a tested object;
  - IS performance characteristics measurement principles and methods at the system points of collection of performance characteristics.

<u>The aim of this work</u> is to develop performance testing technology for wide class of systems for business process automation, including banking systems. Business process automation systems are characterized by requirements on business processes implementation within due time frame. Technical specifications (orders) on these systems contain a set of non-functional requirements, such as productivity, concurrent end users number, restrictions on the holding time of transactions in the system, or responsiveness, etc., that is, explicitly set expected performance properties (performance characteristics) of information system.

In the Second Section of this paper there is suggested IS performance testing type classification and IS main performance characteristics interpretation.

Considering the fact, that performance experiments are rather typical, Section Three proposes four interrelated metamodels, comprising basic concepts and their application beforehand known instructions description (model parameters, basic concepts interrelation schemes in models).

Section Four contains proposed automated performance testing technology description with the use of proposed models and IBM Rational tools as well as tools, invented by the authors. It is shown how to create particular models of tested system with the use of basic concepts and instructions at performance testing planning: requirement model, workload model, system and measurement models, which define all the source data for automated performance testing and expected results. The models allow to automate tool setting of testing automation to the particular performance experiment parameters.

In conclusion, there are outlined major practical results, obtained experimentally in the course of long-term use of banking system performance testing technology.

## 2. Performance test types and evaluated characteristics

An important issue concerning performance testing is to develop common understanding of a problem by both the customer and the contractor (tester). Often, achievement of such understanding turns into a long and tedious process, as the elaboration of common concept base for both sides is required.

Even if a single customer needs the performance testing, the objectives of different experiments might differ as there arise different issues related to performance. Typical problems might be at least as follows:

- Is the hardware in use powerful enough to ensure information system specified productivity over the nearest period (1-2 years) at given workload annual growth?
- Can the information system, which provides the specified throughput rate, at the same time, provide the response time within certain limits under workload increase conditions?
- Are there any signs of degradation of information system after software modification?
- And some other.

In the course of numerous performance experiments execution there is revealed the possibility to typify a number of performance testing problem definitions for the «typical» tasks.

This approach is based on three key elements:

- Performance testing types classification;
- Information system performance characteristics classification;
- The principle of non-destructive control in the course of the performance testing arrangement.

### 2.1. Testing types

Depending on the performance testing objectives, the following types of performance testing can be distinguished:

- Evaluation – performance characteristics evaluation in a single performance experiment;
- Analytical – detection of dependencies (e.g., productivity on computing resources) in a series of performance experiments;
- Configuration – performance characteristics set up and optimization of an information system or its components;
- Regression – multiple periodic performance testing under constant conditions aiming at degradation signs detection of the tested system.

These types of tests are usually carried out as scaled-down experiment. Each type of testing has special features in planning (single experiment, a series of experiments, the need to store experiment results for historical data statistics processing of the performed experiments). At that the scheme of a single experiment is stable enough for assessing each type of information system performance characteristics.

## 2.2. Performance characteristics classification

Elaboration of productivity measured characteristics list plays key role for performance experiment planning, since the conclusions on experiment results are made on the basis of these characteristics values.

The main performance characteristics are responsiveness, productivity and utilization (see table 1).

They accordingly are divided into several calculated (or directly measured) measures; the values of each can be calculated on the basis of directly measured values. Their structure and interrelation under calculation may depend on system engineering platform and the structure of the tested system.

**Table 1. Performance characteristics**

| Performance Characteristics | Calculated measures |
|---|---|
| Responsiveness | Average response time |
| | Average waiting time |
| | Average service time |
| Productivity | Throughput rate |
| | Bandwidth capacity |
| Utilization | Utilization factor |
| | Relative capacity |

**2.2.1. Responsiveness.** Responsiveness is important for system operating in real time, i.e. for systems requiring restrictions implementation within the period of certain tasks or all of the functional tasks solution. In this case information system operational characteristics can be the following: system response time to the user query or problem solution waiting time (for example, the problem on waiting time reduction of business transactions execution can be solved).

The responsiveness characteristics are defined as the time between input data entering and output data acquisition. Responsiveness can be measured both from the point of view of end-user and of computer system. Both business transaction and physical (technical) transaction are subject to measurement. A transaction is a business unit of work implemented to solve a problem for the business. For example, a banking transaction may consist of a double entry transaction payment; thereby the customer is interested in service time not only of the single input, but the entire banking operation. The implementation of banking transactions can be performed by several physical transactions, for example, accesses (references) to the database.

Responsiveness is estimated or calculated on basis of the following performance measures:

– Transaction response time;

– Transaction service time;

– Transaction waiting time.

Response time for a transaction is the time between the transactions initiation and the final execution of the last transaction step.

Response time ($Tr$) in general is made of: service time (the time during which the actual work is done) ($Ts$) and waiting time (the waiting time for resource) ($Tw$):

$$Tr = Ts + Tw.$$

Responsiveness measures depend on the type of system and the structure of its entry and exit points.

**2.2.2. Productivity.** For a large number of systems their integral capacity is important and measured as the number of business transactions processed by the system in a time unit (i.e.., productivity).

In productivity estimation, directly measured or calculated values of the following performance measures are typically used:

– Throughput rate ($V$);

– Bandwidth capacity (or absolute capacity) ($C$).

Throughput rate ($V$) is a measurement of the number of completed transactions per specified time:

$$V_i = \frac{N_i}{T}, \text{ where}$$

$i = \overline{1, n}$ - sequence number of the time period;

$N_i$ - number of completed transactions;

$T$ - period of time.

Bandwidth capacity ($C$) – the maximum number of completed transactions per time unit (maximum throughput rate):

$$C = \max(V_1, ..., \mathrm{V_n}).$$

Productivity characteristics can be used to estimate the system as a whole as well as its parts.

**2.2.3. Utilization.** Specifications of utilization are used to define the extent the tested system resources are used at the given workload.

The following performance measures relate to the utilization:

– Utilization factor (resource utilization);

– Relative capacity.

Resource utilization measures how much a resource delivers service in a timely basis. The general formula is:

$$U = \sum_{j=1}^{m} \frac{1}{\rho_j T_j} \times \sum_{i=1}^{n} \rho_i t_{Si}$$, where

$t_{Si}$ - service time of *i*-transaction

$n$ - the number of serviced transactions;

$m$ - the set of measurement intervals;

$\rho_j$ - resources in the *j*-measurement interval;

$\rho_i$ - resources for *i*-transaction;

$T$ - timely basis.

Resource utilization characteristics and their values depend on the hardware used in the system and its constituent resources: CPU, RAM, external drives, I/O channels, etc.

## 3. Models as properties definition means of performance experiment artifacts

Ensuring the adequacy of the problem definition, the source data and performance results testing require understanding of the key aspects of planned experiment between the customer and the contractor. In a rapidly growing business such understanding should be achieved as soon as possible.

The key aspects of the planned experiments are:

– The problem definition, determining the experiment objective, is to be linked with the system requirements;
– Source data, determining the object of testing and required workload nature;
– Set of required characteristics, which determine test results basis, is to be composed before the experiment is started.

Upon the subject area formalization (for system class) there are formed a few metanotions frames. These metanotions are the metamodels, defining possible concepts, that could be significant with the subsequent performance testing and evaluating the results adequacy.

Source data collection methods to prepare performance experiment, ensuring its adequacy to practical system functioning at the expected workload, are to be defined in the metamodels.

These source data are:

– Information on the performance testing form (assessment, analysis, configuration, regression);
– Information on measured performance characteristics;
– Information on the system structure in the terms of the load feed ways and measurements methods;
– Information on the planned workload structure.

Four metamodels were elaborated, that accomplish selection of required features, performance characteristics and measured values, which adequately characterize tested system functioning process at the performance experiment problem definition:

– Requirement metamodel - characterizes the tested system type and non-functional requirements composition (business rules and technical requirements) of tested system;
– System metamodel - defines the system structure as a queuing systems network (including element composition of “resource” type);
– Workload metamodel - defines system service request number and types as well as service request distribution law during the time of experiment, the service request system entrance rules, service request system entry points (logic level);
– Measurement metamodel – defines composition of characteristics and values collection, the system requests entry points, data collection method and transformation algorithms as well as results evaluation criteria.

In the course of new performance experiment planning using metamodel concepts, models of requirements, system, workload and measurements are shaped by the means of metaconcepts choice and their values measurement, taking into account tested IS characteristics and performance experiment objectives. Making use of metamodels in the course of new performance experiment planning ensures projected models completeness and integrity.

The above said models may vary depending on the types of performance testing and systems.

Metamodels provide a unified approach to the object setting for both the customer and the contractor. They offer such advantages as quick understanding and experiment objectives agreement, quick elaboration of ways to achieve the above mentioned objectives and common comprehension of ways to achieve them. Metamodels are the link between informal requirements of customers with formal definition of the performance experiment in the form of models. This enables to significantly simplify the performance experiment planning and execution automation.

### 3.1. Requirement metamodel

Requirement metamodel encloses requirements formalization rules to system operational characteristics. Such requirements do not contain information on the functions performed by the system and, therefore, they are called non-functional.

Non-functional requirements may be restrictions, defined by the organization business rules, according to which the system operates. Restrictions can specify the time limits of various processes implementation in the system. System capacities on compliance with the time restrictions are directly linked with system throughput rate.

Depending on the specified non-functional requirements, performance experiment objectives are composed and measured characteristics are selected.

Requirement metamodel is intended to define system non-functional requirements. Requirement metamodel can be presented as follows:

$$R = B \cup T \text{, where}$$

$R$ - the set of requirements for the system;

$B$ - the set of business rules;

$T$ - the set of technical requirements.

Business rules include or relate to technological processes, corporate regulations, policies, standards, legislative acts, intra-corporate initiatives, accounting practices, computing algorithms, etc.

A technical requirement pertains to the technical aspects that one's system is to perform, such as performance-related issues, reliability issues, and availability issues.

Requirement metamodel specifies definition rules of verbal requirement model, containing system non-functional requirements.

A tested system particular requirement model is actually formed according to the metamodel upon the choice of specific concept values or rules.

## 3.2. System metamodel

The system metamodel enables to define the system structure as queuing systems network, consisting of the «resource» - type elements and links between them.

The system metamodel holds a complicated structure and determines the object of testing formation rules, which is defined up to the level of the devices involved in the testing and software (components, services) with certain performance characteristics.

The metamodel presupposes that a certain class of systems can be represented in the form of certain package of some concepts and their interrelation rules.

A particular tested system model is projected according to metamodel upon the choice of specific concept values and rules.

The system metamodel can be represented as follows:

$$\sigma = \{\{U(p)\}, \{S\}, K_S, K_U\} \text{, where}$$

$\{U(p)\}$ - the set of devices of the object of testing with the performance characteristics;

$\{S\}$ - the set of bundled software (components, services);

$K_S$ - bundled software communication matrix, the rows of which are the sources, the columns are the receivers, and the cells indicate links availability between the latter two;

$K_U$ - software systems and devices link matrix, which characterizes the number of resources involved in the device for software system. The matrix rows are software systems, the columns are the computer systems. The matrix elements are the dedicated resources vectors.

Load feed and characteristics collection points tend to be any communications in the matrices $K_S$ and $K_U$, which are particularly defined in the workload and measurement models.

The tested object definition rules are rather complicated. They are divided into software definition and hardware, as well as communications between them.

System specification level as tested object is determined by the experiment objectives. It can be either a single software system (the system as a whole), or a set of software packages or applications (modules).

The software definition is based on the principle, that the tested system is considered as a black box with multiple inputs and outputs. Depending on the needs, the system can be decomposed into software systems, which are also regarded as a black box with multiple inputs and outputs, and that can communicate to each other. Particular blocks communication can be defined both as load feed points and characteristics collection points.

Hardware definition is to include hardware static characteristics, such as number of processors, memory, and dynamic characteristics, for instance, resources dynamic reallocation rules. These rules determine sharing opportunities of certain resources, the priorities of the resources usage by some software system and weighting factor of corresponding resources allocations to software systems.

## 3.3. Workload metamodel

The workload metamodel determines the input workload flow structure.

The workload metamodel can be represented as follows:

$$L = \{F, M, I\} \text{, where}$$

$F$ - the functions set, defining the input workload distribution;

$M$ - the multi-dimensional matrix, dimensions of which may be workload types, such as workload flow sources, flow names and types, and the elements of which are their quantities;

$I$ - the interfaces set for the workload input (as a link to the system model).

The workload flow is structured as per its dynamic and static properties.

Dynamic properties of this workload imply their distribution laws in the course of time. The distribution laws can be:

– Deterministic;
– Probabilistic.

Deterministic workload distribution laws in the course of time imply scheduled given workload entrance into the system. In extreme case the schedule can include the arrival time of each request into the system. Probabilistic distribution laws of workload in the course of time imply indication of normal, uniform, exponential, or other law distribution.

The workload flow quantitative composition determines its static properties. The workload flow size can be structured as per some criteria, including:

– workload type;
– sender type.

For the considered system class there are three basic workload types, namely:

– traffic;
– messages;
– events.

The workload as traffic is created by users in a system built on client-server architecture. The most common types of traffic are HTTP-and SQL-traffic. The actions of users at the automated workplace cause the queries formation (e.g., SQL-queries) to the server. In every system there are different types of automated workplaces - requests senders.

Another workload type is the workload in the form of messages. Messages are used as information exchange units between parts of bigger systems. Depending on the destination, messages can be divided into several types. Messages containing documents for processing, in their turn, can be divided into types according to their formats. In addition, messages can be:

– single (containing a single document);
– package (containing a set of documents).

Depending on the sender, each message can be encoded and signed with one or more electronic digital signatures.

There is workload created by the tested system as the result of different events. The latter include the implementation of certain regulatory procedures executed by either a schedule or an operator.

Particular tested system workload model is shaped according to the metamodel at the choice of concepts specific values or specific rules. The workload model is defined at the planning stage of the experiment.

## 3.4. Measurement metamodel

The measurement metamodel is intended to unify the definition of:

– Ways to get the measured values in the process of the information system performance testing;
– Fundamental opportunities of measurement process statement;
– Typical evaluation methods of measures and characteristics;
– General tools properties to analyze their use opportunities for measurements automation.

The measurement metamodel can be represented as follows:

$$\Delta = \{\{|U|\}, \tau, \mu, R, \omega\}, \text{ where}$$

$\{|U|\}$ - the measured quantities list for each type device of information system;

$\tau$ - measurement frequency and off-duty factor;

$\mu$ - the set of estimates and their interrelation with the measured values;

$R$ - estimates obtaining standard rules and algorithms;

$\omega$ - standard criteria of obtained results evaluation.

Specific concept values or specific rules choice is carried out at the stage of performance experiment planning, that is, actually specific measurement model is shaped according to the metamodel.

## 3.5. Models Interrelations

The presented models are interrelated and closely interact with each other (see figure 1). All of the four models are to be defined for every experiment.

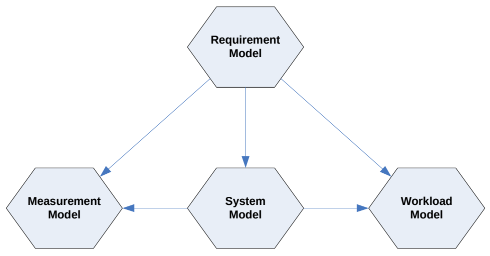


**Figure 1. Models interrelation**

The requirement model is the initiating model.

After testing objectives formation, the following can be defined:

– Characteristics and indices to be defined and criteria they are to comply with - a <u>requirement model</u>;
– Object of testing: the part of the system subject to test - a <u>system model</u>;
– System requirements workflows from a controlled process - a <u>workload model</u>;
– Parameters to be measured and points of measurement to obtain required results - a <u>measurement model</u>.

Some measurement model characteristics and workload model can be defined only after the completion of the system model definition. These characteristics, for example, include a definition of the load feed points and the characteristics collection points.

## 4. Model-based performance testing technology

The Model-based performance testing technology consists of the following stages:

– Testing objectives definition;
– Program and testing methods development;
– Preparation for testing;
– Load feed;
– Data collection;
– Results interpretation and analysis.

Figure 2 shows the technology of performance testing, highlighting automated operations.

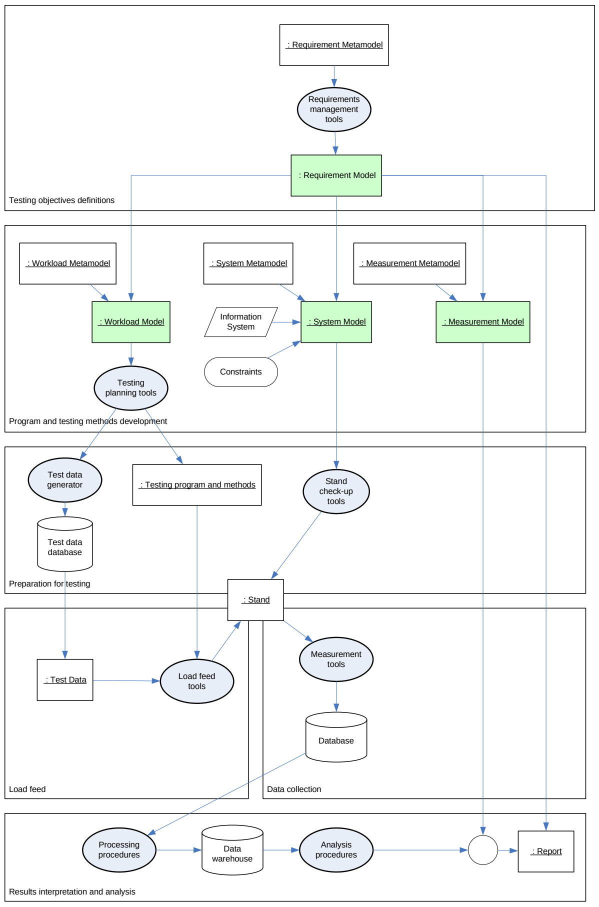


**Figure 2. Model-based performance testing technology (automated operations are highlighted)**

At the "Testing objectives definition" Stage the requirement formalization rules for system performance characteristics are defined and the *requirement model* is projected with the help of the *requirement metamodel.*

Performance testing objectives and performance testing scenarios are agreed with the customer making use of the requirement model.

At the "Program and testing method development" Stage scenarios definitions, performance testing objectives and tested performance characteristics requirements are added into "Testing program and methods" Document from the *requirement model*.

At the above said stage the *workload metamodel* is used to determine static and dynamic possible structure as well as workload flow composition. The *workload model* is shaped on the agreement basis of the workload flow composition and scripts with the customer making use of the *workload metamodel.*

*System metamodel* enables to define the system structure in the form of the *system model* as queuing system network. *System model* contains performance testing stand requirements, that are to be agreed with the customer.

The set of the required tested characteristics and calculated indices, measured in the course of the experiment, performance methods and testing result evaluation criteria are documented in the *measurement model* on the basis of the *measurement metamodel*.

At the "Preparation for testing" Stage the setting of testing planning tools and of testing data generator is performed on the basis of the *workload model.*

Test data are generated automatically in accordance with the qualitative workload type composition, specified in the *workload model*. Generated test data are stored into the test data database.

The stand check-up tools and measurement automation tools setting is performed according to the *system model* in the unit of data collection points and data collection devices involved. The setting of measurement automation tools in the unit of collected characteristics list is performed according to the measurement model.

At the "Load feed" Stage the automation tools perform load feed procedures at entrance points, specified for each workload type in the *system model.* The workload is extracted from the test data database, provided in advance, and is fed automatically as per the schedule (the law of distribution in the course of time), specified for each workload type in the *workload model*.

At the "Data collection" Stage the automated collection of the tested characteristics values is performed. The measurement devices get the values collection points from the *system model*, and measured characteristics composition – from the *measurement model*.

At the "Result interpretation and analysis" Stage all of the four models are used by the automation tools.

The *requirement model* is used for the experiment result compliance to the system requirements. The *workload model* and the *system model* provide the shaping of the report on the experiment performance conditions. The experiment result comparison to the evaluation criteria is carried out in the automated mode on the basis of the *measurement model*.

A more detailed description of each stage is given below.

## 4.1. Testing objectives definition

The testing objectives are defined on the customer need basis for evaluation or predicting of IS operational characteristics and corresponding non-functional requirements.

The customer is not always able to define the testing objectives adequately. In the process of the main objective agreement and upon the program and testing methods development, some secondary testing objectives and restrictions can be detected, that are also to be agreed. The developed requirement metamodel can help properly inquire the customer at an early stage and adequately define the testing objectives. The above mentioned reduces the number of the objectives agreement alterations as well as the number of objectives setting together with the summarized time allocated for determination of main and secondary performance testing objectives.

**4.1.1. Requirement model definition.** At the stage of testing objectives defining using requirement metamodel, the tested object is analyzed and the primary requirement model is shaped, which is later discussed with the customer, edited and approved.

The requirement metamodel represents a verbal definition of non-functional requirements. These requirements are grouped in conformity with the types and objects, to be applied to. This model also contains criteria for evaluation of characteristics and calculated indices obtained as the result of performance testing.

The IBM Rational RequisitePro tool is used for the requirement model shaping automation.

## 4.2. Program and testing methods development

This is one of the most important ant difficult stages of performance testing. At this stage the system model, workload model and measurement model are developed on the basis of the corresponding metamodels.

The three models are developed simultaneously and are closely connected with each other. For example, the load feed interfaces definition and characteristics collection points definition are possible only after the system model is defined.

Upon completion of this stage all the approved models are included in the document "Testing program and methods" that represents the plan of performance testing and is approved by the customer.

**4.2.1. System model development.** At the stage of the program and methods development, according to the objectives set, verbal and graphical models of the system are developed, which is later discussed with the customer, edited and approved.

System model represents the definition of the tested object, which includes software and hardware, their connections and allocated resources (storage, processor etc.).

Figure 3 shows an example of system model in the form of a graphical scheme of software and computing systems interrelation.

Matrices $K_S$ and $K_U$ for the presented scheme look as follows:

$$K_S = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}; \quad K_U = \begin{pmatrix} p_1 & 0 & 0 & 0 \\ 0 & p_2 & 0 & 0 \\ 0 & 0 & p_3 & 0 \\ 0 & 0 & 0 & p_4 \\ 0 & 0 & 0 & p_4 \end{pmatrix}, \text{ where}$$

$p_i$ - is the vector of performance characteristics, that defines the number of resources allocated in a computer system for a certain software system.

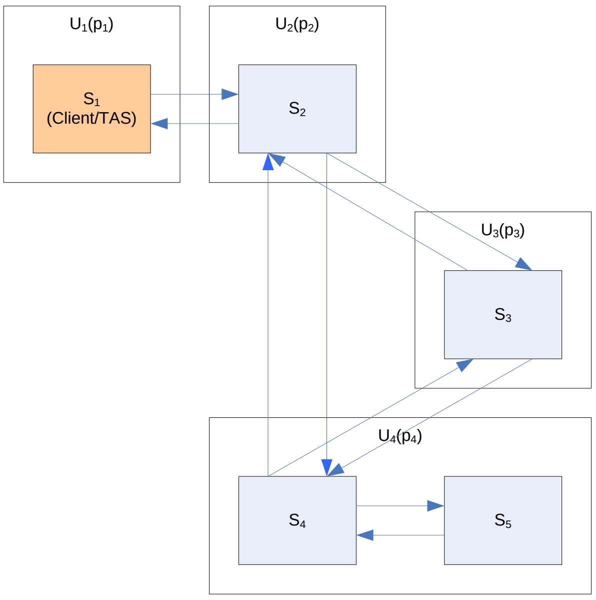


**Figure 3. An example of a system model in the form of a graphical scheme**

**4.2.2. Workload model development.** At the program and methods development stage the workload model is developed according to the set objectives, which is later discussed with the customer, edited and approved.

The IS workload values (extreme, as a rule), expected by the customer, and set of workload sources, which is to be emulated during the tests, are to be considered in the model. In fact, the system workload scenario corresponding to performance testing objectives is developed in co-operation with the customer. The workload model with all the workload parameters defined is developed as a result of the scenario detailed study.

The workload model represents the definition of test data and their entrance rules into the system. In fact, source data for tool-generated workload are formed according to the workload model.

**4.2.3. Measurement model development.** At the stage of the program and methods development according to the set objectives, verbal measurement model is developed, which is later discussed with the customer, edited and approved.

The measurement model represents the definition of the collected characteristics, methods of their collection and interpretation.

Methods and means of the measured parameters collecting values are defined in the measurement model. It is very convenient to use the measurement metamodel in which measurements standard mechanisms are defined, usually supported by the tools available in the market. Values collection methods of measured parameters can strongly differ depending on the set testing objectives.

Generally these methods can be subdivided into methods of non-destructive control, methods of destructive control and mixed methods.

*In case of the non-destructive control system* tools of information system operational environment and automated testing tools are used for collection of the measured parameters values. *In case of the destructive control* some part of measuring tools is integrated into applied software of the information system. *In case of the mixed control* integration of measuring tools is possible both into the system tools and into applied software.

At best measurement tools are not to influence the tested system. In cases when it is necessary to take advantage of the measurement tools, which influence tested system (set interruptions and trigger events) separate tests are performed in order to evaluate a level of measurement tools influence on the tested object performance. The obtained values of influence level are taken into account further at interpretation of results.

## 4.3. Preparation for testing

Preliminary accomplishment of the following operations is required for carrying out of the performance testing:

– Preparation of stand;
– Preparation of test data;
– Preparation of resources for the load feeding;
– Preparation of the measurement tools.

Preparation of the test stand is carried out according to the approved system model and includes organization of technical equipment and software and ensuring of additional restrictions performance, for example, time synchronization on all installations of the tested object, etc.

Checking of the test stand based on the system model is automated. Errors during stand preparation can lead to the failure of the performance experiment, as dysfunction of the stand parts can be detected after the beginning of the experiment. Moreover, incorrect customizations of the stand can distort the results of performance testing that can lead to necessity of repetition of the experiment. The existing formalized system model has allowed automation of the stand check, having reduced the time of such check. When errors in configuration of the stand and its technical equipment and software are detected, automated check can be repeated many times at low labor input due to the process automation.

Test data for the performance testing are prepared according to the approved workload model. Test data are either generated or are real data that meet the characteristics according to the workload model used.

The planning of performance experiment and generation of test data is automated on the basis of the workload model. Planning time and quantitative parameters of test data flow, as well as the load feed script are defined in the course of testing. The automation tool of performance experiment planning allows entering appropriate input data on forthcoming test into previously defined screen forms. On the basis of this information the automated generation tool prepares the test data. The automated preparation of the test data provides their complete correspondence to a current state of a tested system. Thus, it is guaranteed that all test data are entered into the system and are processed the same way as the real ones.

Workload feed interfaces (and appropriate tools) are customized according to the system model.

By preparation of measurement tools according to the measurement model, suppliers and receivers are customized.

The result of the preliminary operations stage is the test stand ready for testing (being in an initial state) and test data ready for feed.

## 4.4. Load feed

At the stage of the load feed the prepared test data are fed into the tested system according to the laws of their allocation defined in the workload model.

The stage of the load feed is automated using testing tools that can send different workload types in accordance with the distribution laws. The workload in the form of messages filed from a prepared base of test data. The workload as traffic may be submitted in the form of HTTP-, and SQL-traffic. The composition of the traffic is generated on the basis of automated analysis of queries code, which is sent by automated workplaces of the tested system by execution of different operations of any user. The workload as some events is formed through emulation of the actions, which the operator performs. The emulation is executed thought testing tools.

## 4.5. Data Collection

The data collection is carried out by the tools of measurement. The basic information on the values of measured indices is automatically collected. This information includes such indices as loading of CPU, use of memory, time of incoming transaction into the system, the system response time, etc. All of the data are stored in the operational database for further processing.

Depending on the methods of collection defined in the measurement model, the data can be collected both in the course of testing as they arrive, and after testing performance. The first case is preferable because the data collected in process of arrival can be used for the control of testing process performance.

Data collecting lasts until the process of testing meets the criteria of completion and moves to the next stage, defined in the measurement model.

The data preprocessing also depends on the way of processing defined in the measurement model and can be fulfilled both in the process of the data arrival, and after the experiment applied to the whole collection of the gathered data.

## 4.6. Results interpretation and analysis

The obtained testing results are interpreted according to the measurement model for further analysis: metrics calculation and values evaluation of the IS operating characteristics are carried out.

Based on the *measurement model* the multi-step automated processing of raw data is performed. At first, measurements on the basis of rules and algorithms of the measurement model are formed. Estimated measurements are stored in a data warehouse for subsequent analysis. Based on the data warehouse the analytical reports are produced containing estimates of the values and criteria for evaluation in accordance with the measurement model and requirement model. Due to the presence of the repository it is possible to analyze not only results of the finished performance experiment, but also to perform a comparative analysis of several regression tests.

On completion of the results interpreting, conclusions concerning the compliance of the IS to the objectives are made, taking into account the restrictions and criteria of the requirement model. The test results in practice are drawn up in the form of the protocol of performance testing.

## 5. Conclusion

The described technology has been applied for four years. It was used for testing of some banking systems built on different platforms: Windows, Linux, z/OS. Up to forty applications and data bases functioned simultaneously in a computing system in various experiments on performance testing. The data base maximum volume comprised 2.5 TB. Such experiments, carried out at the customer site required a lot of manual activity. The experiments ware accompanied by the recording of the available workload flows.

To carry out adequate testing to evaluate the IS expected characteristics on the short-time basis was technically impossible, as rather low man-hour rate was involved (5-10 men/months).

The situation evaluation with the methods employed was extremely tedious and in the majority of cases was not performed. Systematical anticipatory monitoring of performance characteristics degradation was not carried out in the course of software modification due to the high labour intensiveness of the above mentioned works.

The volume of the experimental work over the last 4 years is as follows (Summary):

- Performance experiments amount: more than 40;
- The number of generated messages: more than 75 million;
- The average number of messages in a single experiment: about 2 million.

Table 2 reflects the data on the average duration of work in the past and now.

**Table 2. Duration of work stages**

| Stage | Duration | |
|---|---|---|
| | in the past | now |
| Testing objectives definition | 2 week | 1 week |
| Program and testing methods development | 5 weeks | 2-3 days |
| Preparation for testing | 3 weeks | 2 day |
| Load feed<br>Data collection | is dependent on the experiment plan | |
| Results interpretation and analysis | 1 week | 3-4 hours |

The table shows that the developed technology and tools of its automation make it possible to prepare and conduct the performance testing within a reasonable time frame for the customer. The number of repeated experiments for obtaining the required characteristics has decreased from 2-3 to 1 experiment.

During the testing of major systems, working on expensive technology, it is important to minimize the risks such as shift in time schedule of experiments performance, repetition of experiments, etc.

Such risks occur, usually, due to the lack of common understanding of the performance testing objectives by the customer and the contractor, and errors in planning of the experiment.

Applying the developed technology, it is possible to reduce such risks, and lower labor intensity and duration of performance testing, and hence to reduce their cost.

The technology of automated performance testing arrangement and performance is elaborated in this work and it is intended for the IS class. The technology is based on the use of the metamodels, that define requirements to the performance experiment performance as well as the tested object properties and its workload. Within the framework of the technology the models are created step-by-step on the basis of the metamodels. The above said metamodels are comprehensible for the customer and serve as foundation to agree the experiment main parameters and result evaluation criteria. The models assist in tools setting, after which the experiment is performed automatically under the guidance of the performance testing tools. The technology covers all the planning aspects, performance experiment execution and its result analysis. The models use considerably (by several times) reduces performance testing labour intensiveness compared to the current practice and enables to repeat the prepared experiment, for example, in case of IS degradation control in its life cycle.

## 6. References

[1] Kostogryzov A., V.Panov, B.Pozin, V.Sablin. Mathematical modeling of processes in systems life cycles in compliance with standards requirements of ISO/IEC 15288 and ISO/IEC 12207, Spincose, Montreal, Canada, 2003.

[2] Kozlov A.N., Pozin B. A., Banking system complex load testing technology. International Scientific and Practical Conference «Business Processes Re-engineering on the basis of modern knowledge management system IT», 2006 (RBP-SUZ-2006), Moscow, pp.130-131.

[3] ISO/IEC 1539 “Information technology – Software engineering – Software measurement process”

[4] Prof. Flavio Oliveira. Performance Testing: an Introduction FACIN-PUCRS, 2008.

[5] Ibrahim K.El-Far and James A.Whittaker. Model-based Software Testing. Florida Institute of Technology. 2001.

[6] Alexandre Petrenko: Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography. MOVEP 2000.

[7] Kim G.-B. F method of generating massive virtual clients and model-based performance test/ Fifth International Conference on Quality Software, 2005, pp.250-254

[8] Changyou Xing, Guomin Zhang, Ming Chen. Research on universal network performance testing model/ International Symposium on Communications and Information Technologies, 2007, pp.780-784

[9] ГОСТ 34.602-89. ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы

# Inheritance of Automata Classes Using Dynamic Programming Languages (using Ruby as an Example)

*Kirill Timofeev*
*SPbSU IFMO*
email: ktimofeev@dataart.com

*Artyom Astafurov*
*SPbSU IFMO*
email: astaff@dataart.com

*Anatoly Shalyto*
*SPbSU IFMO*
email: shalyto@mail.ifmo.ru

## Abstract

*This paper analyzed two libraries for implementation of automata in dynamic languages. The comparison has been made based on modifying the functionality of the Restful-authentication plugin for Ruby on Rails framework.*

*These libraries allow transferring a graphical model into a source code. Moreover library developed by the authors of the paper provide a method for creation of state groups which reduces the number of transitions required to implement the automaton in the code. Also this library preserved the hierarchy of the parent automata after inheritance. Using developed library it is possible to perform a reverse engineering and restore the original graphical model from the source code.*

## 1. Introduction

Year over year the dynamic programming languages are used more often in software development. For example, in 2008 the proportion of dynamic languages to the languages with static type checking was 40% [1]. Dynamic languages allow for runtime program extension by dynamic creation of new methods and usage of macro scripts [2]. *Ruby* is a dynamic programming language and is a part of top 10 most popular languages in 2008 [1]. Also this language has been used to develop a popular open source web-application framework *Ruby on Rails*. One of the features of *Ruby on Rails* is a flexible functionality extension mechanism that uses plugins that can be added into *Ruby on Rails* application. *Restlful-authentication* [4] is one of such plugins that allows for web application to support user registration functionality. This plugin is used in 96% [5] of applications developed on *Ruby on Rails* and is implemented using *Automata* approach.

*Restful-authentication* plugin uses *Acts as State Machine* [6] library that doesn't allow preserve the hierarchy of the parent automata when the automaton is inherited and doesn't have nested groups of states concept. This leads to duplicated code, makes debugging more complicated and also doesn't allow an isomorphic generation of the model based on code when necessary. In paper [2] a method for inheritance of automata classes has been proposed, also a *State Machine on Steroids* library has been developed to support the concept. This library uses the traditional object-oriented paradigm applied to automata and extended with some features of dynamic libraries, which means that automata classes are created in runtime. Using the method proposed by authors of paper [2] will help to eliminate the drawbacks of the *Restful-authentication* plugin implementation.

The goal of this research paper is to compare the approaches to inheritance of automata classes described above using *Restful-authentication* plugin implementation as an example.

## 2. Graphical Notation Being Used

As a graphical model it is proposed to use state diagram from *UML 2* [7] extended with graphical notation for inheritance of automata classes. This notation has been proposed in [8], as in *UML 2* it is impossible to present inheritance of automata classes. The sample use of an extended graphical notation is shown on fig.1.

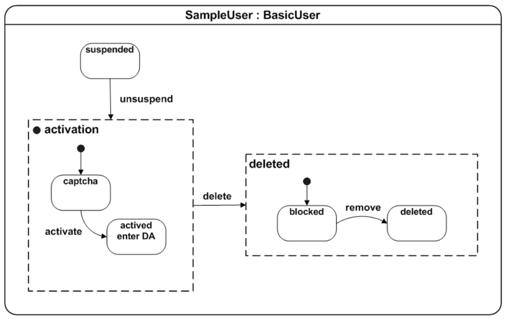


**Fig. 1.** Extended Graphical Notation

On this figure the automaton class `SampleUser` is presented. This class is inherited from automaton class

`BasicUser` and contains the following changes to its base class:

- added a new group `deleted` which is made of two states: `blocked` and `deleted`. The initial state is set to "`blocked`";
- the group activation has been overridden. It now contains `captcha` state which is marked as initial. A transition from this group to the new group `deleted` as been added;
- new state `suspended` has been added. There is a transition from `suspended` state to `activation` group.

## 3. Problem Statement

Let's take *Restful-authentication* plugin and review it in more details. Assume, for example, that in order to register successfully the user has to take three steps:

1. Fill in the registration form on the website;
2. Receive an e-mail with activation code;
3. Confirm the registration by entering the activation code into a special form on the website.

In case the user hasn't registered in a given period of time or entered the activation code incorrectly several times, he will be blocked and won't be able to register on the site any longer. The site administrator can block the users as well as delete from the system.

On fig. 2 using the extended graphical notation the automaton for basic user registration (`BasicUser`) is presented.

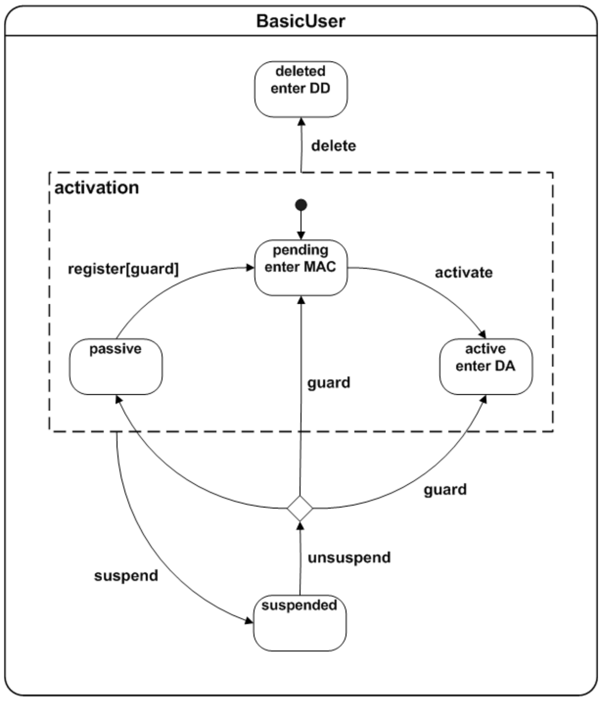


**Fig. 2.** Basic user registration

It consists of `activation` group and `suspended` and `deleted` states. The nested group `activation` contains three states: `passive`, `active` and `pending`, the latter is marked as initial.

The `AdvancedUser` automaton (fig. 3) is inherited from `BasicUser` automaton and has the following functionality:

1. After entering the activation code the user should perform two additional actions: recognize Captcha (to avoid spam registrations) and submit user data on a profile page. In order to do this, a nested group `active` should be created.
2. The deletion of the user is performed in two stages: on the first stage the user is blocked, but his messages are still being stored in the system, on the second stage both: the user and his data are deleted. In order to do this a nested group `deleted` should be created.

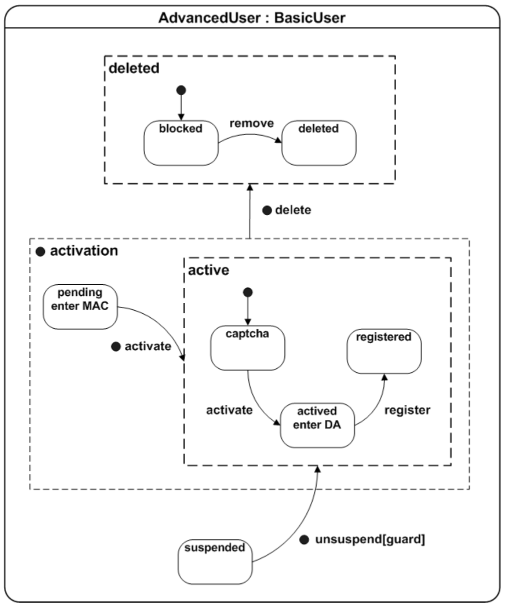


**Fig. 3.** Extended User Registration Automata

## 4. The Implementation of Inheritance using Act as State Machine Library

Let's look at the `BasicUser` class that represents a basic user in *Restful-authentication* plugin using *Acts as State Machine* library. It provides four methods:

- `acts_as_state_machine :initial` – sets the initial state of the automata;
- `state` – creates the state. As optional parameters it accepts lambda-functions [2]: the actions to be done when entering or leaving the state;
- `event` – named transition;

- `transitions` – defines from/to states for making a named transition. As an optional parameter the method accepts a lambda-function that can contain a guard condition.

This library doesn't support nested groups. In this case in order to make an isomorphic transfer of a diagram with nested groups into the code it is required to modify the original `BasicUser` diagram as shown on fig. 4 by "flattering" it:

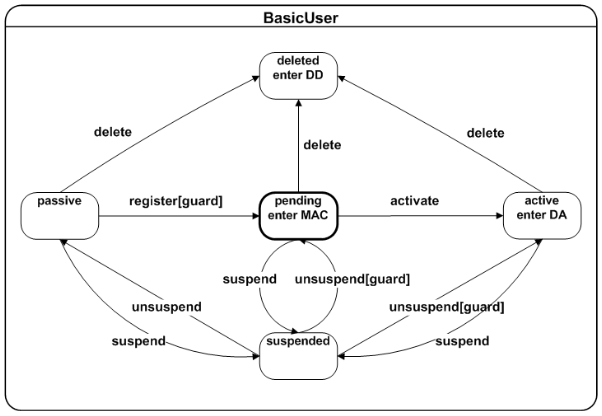


**Fig. 4.** A Basic User Registration Automata Without Nested Groups

Below is a *Ruby* code snippet that implements the model described above:

```
class BasicUser
  acts_as_state_machine :initial => :pending

  state :passive
  state :pending, :enter =>
:make_activation_code
  state :active,  :enter => :do_activate
  state :suspended
  state :deleted, :enter => :do_delete

  event :register do
    transitions :from => :passive, :to =>
:pending,
                :guard => Proc.new {|u| !
(u.crypted_password.blank? &&
u.password.blank?) }
  end

  event :activate do
    transitions :from => :pending, :to =>
:active
  end

  event :suspend do
    transitions :from => [:passive,
:pending, :active], :to => :suspended
  end

  event :delete do
    transitions :from => [:passive,
:pending, :active, :suspended], :to =>
:deleted
  end

  event :unsuspend do
    transitions :from => :suspended, :to =>
:active,
                :guard => Proc.new {|u| !
u.activated_at.blank? }

    transitions :from => :suspended, :to =>
:pending,
                :guard => Proc.new {|u| !
u.activation_code.blank? }
    transitions :from => :suspended, :to =>
:passive
  end
end
```

Let's create a new class `AdvancedUser` which is represented on the fig. 5. This class will be inherited from `BasicUser` class. The new class will inherit the internal structure of the `BasicUser` automaton created using the methods of *Acts as State Machine* library. Since the library doesn't support nested groups the hierarchy of the parent `BasicUser` automata will be lost in the inherited `AdvancedUser` automata.

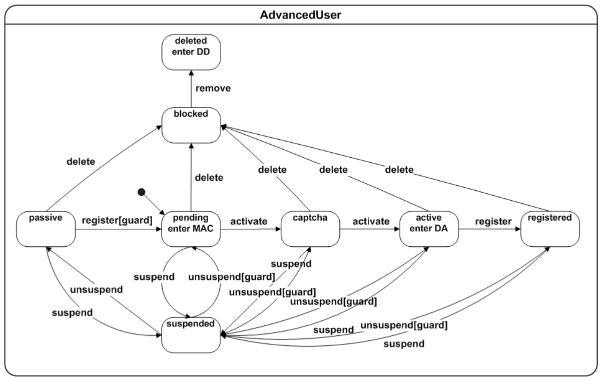


**Fig. 5.** The Model of the `AdvancedUser` Automata Class After Losing the Hierarchy of the Parent Automata

Below is the code that implements the `AdvancedUser` automaton:

```
class AdvancedUser < BasicUser
  state :captcha
  state :registered
  state :blocked

  event :activate do
    transitions :from => :captcha, :to =>
:actived
    transitions :from => :pending, :to =>
:captcha
  end

  event :register do
    transitions :from => :activated, :to =>
:registered
  end
  event :unsuspend do
    transitions :from => :unsuspend, :to =>
:captcha
  end

  event :suspend do
    transitions :from => [:passive,
:pending, :active, :suspended, :captcha,
:registered], :to => :suspended
  end

  event :delete do
    transitions :from => [:passive,
:pending, :active, :suspended, :captcha,
:registered], :to => :deleted
  end
end
```

The approach described above has the following drawbacks:

- The loss of hierarchy of the parent automata. This results the inability to do the isomorphic transfer of the code into the diagram preserving the structure of the original diagram;
- No support for nested groups of states which results in increase of the number of transitions in the automaton.

## 5. The Implementation of Inheritance Using State Machine on Steroids Library

In order to eliminate the drawbacks of the described above approach, the author of this paper have implemented a new library called *State Machine on Steroids*. Like *Acts as State Machine* it uses dynamic programming language to implement automata classes. The library provides the following six methods:

- `automaton` – the method accepts the block as an argument. The block has a *DSL*-syntax (*Domain Specific Language*) designed to describe the automaton [2];
- `group` – creates the nested group of states;
- `initial` – sets the initial state for the automaton;
- `state` – creates the state in the automaton in runtime. As an optional parameter it accepts a lambda-function that represents an action on entering/leaving the state;
- `event` – named transition;
- `transitions` – defines the starting and the destination states for the named transition. As an optional parameter the method accepts a lambda-function that serves as a guard condition for the transition.

Let's implement the automaton presented on fig. 2:

```
class BasicUser
  include StateMachineOnSteroids

  automaton :user, :initial => :activation do
    state :deleted

    group :activation, initial => :pending do
      state :pending do
        transition :activate, :to => :actived
      end

      state :passive do
          transition :register, :to =>
:pending,
                  :guard => Proc.new {|u| !
(u.crypted_password.blank? &&
u.password.blank?) }
      end

      state :activated do
      transition :suspend, :to => :suspended
      transition :delete, :to => :deleted
    end

    state :suspended do
      event :unsuspend do
        transition :to => :active,
                  :guard => Proc.new {|u| !
u.activated_at.blank? }
        transition :to => :pending,
                  :guard => Proc.new {|u| !
u.activation_code.blank? }

        transition :to => :passive
      end
    end
  end
end
```

In code above every call to the method of the library creates a separate class. This approach preserves the structure of the parent automata `BasicUser` after inheriting it.

Let's implement the inherited class `AdvancedUser` (fig. 3):

```
class AdvancedUser < BasicUser
  automaton :user do
    group :deleted, initial => :blocked do
      state :blocked do
        transition :remove, :to => :deleted
      end

      state :"user::activation::deleted" do
    end
    group :activation do
      group :active, initial => :captcha do
        state :captcha do
          transition :activate, :to =>
:activated
        end
        state :"user::activated" do
          transition :register, :to =>
:registered
        end
        state :registered
      end
    end
    transition :"user::delete", :to =>
:deleted
    transition :"user::activate", :to =>
:actived
    state :suspended do
      event :"user::unsuspend", :to => :active
    end
  end
end
```

Using the described above *State Machine on Steroids* library we were able to eliminate the drawbacks of *Acts of State Machine* library, discussed in Part 3 of this paper:

- a method for creation of state groups has been introduced which reduces the number of transitions required to implement the automaton in the code;
- the hierarchy of the parent automata is preserved after inheritance. This allows to reference to the previously created classes, for example we can reference `:"user::deleted"`. Also this helps to solve the problem of automatic isomorphic transfer of the source code back to the graphical model.

## 6. Side-by-Side Comparison of Acts as State Machine and State Machine on Steroids Libraries

*State Machine on Steroids* library retains all the advantages of the dynamic and object-oriented features of *Ruby* programming language, such as:

- the ability to create the code using domain specific language. As a result, the domain experts will be able to easier understand, verify and modify the source code;
- the support for the development of self-documenting code;
- increase of the quality, reliability and maintainability of the programs;
- preservation of the hierarchy during the inheritance of the automata.

On a special note, one of the advantages of the *State Machine on Steroids* library is the ability to perform an isomorphic transfer of the graphical notation into the source code and vice versa. This also allows to avoid duplicate code that is produced as a result of the implementation of group transitions in *Acts as State Machine* library.

The number of transitions and states required to implement the automaton displayed on the fig. 3 is shown in the table below for the libraries described above. Please note that *State Machines on Steroids* library simplifies the code making it less redundant.

**Table.** The Comparison of Libraries

| Library | Transitions | States | Groups |
|---|---|---|---|
| **Acts as State Machine** | 20 | 8 | - |
| **State Machine on Steroids** | 9 | 8 | 3 |

## 7. Conclusions

In this paper we have analyzed two libraries for implementation of automata in dynamic languages. The comparison has been made based on modifying the functionality of the *Restful-authentication* plugin.

These libraries allow for transferring a graphical model into a source code. At the same time a well-known *Acts as State Machine* library doesn't preserve the hierarchy of the parent automata after inheritance and doesn't support isomorphic implementation of nested groups of states.

The *State Machine on Steroids* library developed by the authors of the paper eliminates the drawbacks listed above. Also using this library it is possible to perform a reverse engineering and restore the original graphical model from the source code.

## 8. References

[1] TIOBE. "TIOBE Software: Tiobe Index". http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[2] Timofeev K. I., Astafurov A. A., and Shalyto A. A. "Automata classes inheritance with dynamical language Ruby" / Software Engineering Conference (Russia) 2008. http://www.secr.ru/?pageid=4548&submissionid=5270

[3] David H. H. "Ruby on Rails". http://www.rubyonrails.com/

[4] Grant G. "Restful Authentication Generator". http://github.com/technoweenie/restful-authentication/tree/master

[5] Szinek P. "Rails Rumble Observations, part II – trends in gem/plugin usage". http://www.rubyrailways.com/rails-rumble-observations-part-ii-trends-in-gemplugin-usage/

[6] Scott B., "Acts as State Machine". http://agilewebdevelopment.com/plugins/acts_as_state_machine.

[7] "Object Management Group. Official UML Specification". http://www.uml.org/#UML2.0

[8] Shopyrin D. G., and Shalyto A. A., "Graphical notation for automata classes inheritance" // Programming. 2007. #5, pp. 62–74. http://is.ifmo.ru/works/_12_12_2007_shopyrin.pdf

# Automata-based Programming in Visual Studio 2005: State Machine Designer Tool

Evgeny O. Reshetnikov
*Saint-Petersburg University of Informational Technologies, Mechanics and Optics*
*ereshetnikov@ rambler.ru*

## Abstract

*This article introduces State Machine Designer tool for automata-based programming in Visual Studio 2005. State Machine Designer extends functionality of Visual Studio 2005 and gives developer more abilities for designing and realization of software products. This tool allows developer to create UML-like visual models of project, add automata behavior to any class in the project, and generate a part of source code on C# programming language.*

## 1. Introduction

Currently it becomes obvious that there is no common way in developing of software products because of many different programming standards and techniques. Programming requires more and more specifications to make this process clear. Specification is a good practice but it is not enough because it does not reflect the code which developer will produce. Usage of visual models is a good way to bring source code concepts and specifications together. Modelling allows developer to see main structure of a project and main interactions between its components. The last problem is to make models and source code being related. *UML*-diagrams [1] may describe both of system behavior and project structure simultaneously. *State Machine Designer* allows usage of three *UML*-diagrams during project developing. Those are *Classes Diagram*, *Objects Diagram* and *State Machine Diagram*. By using these three models a part of source code could be generated automatically.

The following *UML*-diagrams are used in *State Machine Designer*:

*Classes Diagram* – describes classes, interfaces and its relations.

*State Machine Diagram* – describes behavior of the entity whose lifecycle could be represented as a state machine. This state machine consists of finite count of states and transitions between them. States represent some stable states of the system and transitions occur on the specified system events and if corresponding guard conditions are satisfied [2].

*Objects Diagram* – reflects instances of classes with initial values of its properties and aggregation relationships between these instances.

## 2. Automata-based programming

Automata-based programming is a programming technology where finite state machines are used for representing the whole program or some of its parts.

Finite state machines could be implemented using different methods such as state design pattern [3] or SWITCH-Technology [4] or any other well-known method. But the main idea remains the same: introduce finite number of states, provide transitions between these states on some system events and make specified actions on states entering, leaving and transitioning between them.

Automata-based programming is helpful in development of compilers, automation solutions and every application whose logic could be represented as finite state machines.

## 3. Implementation

State Machine Designer tool developed as a plug-in for Visual Studio 2005 [5] and based on Domain-Specific Language Tools [6] which is a part of Microsoft Visual Studio 2005 SDK.

*Domain-Specific Language Tools* are used for creation of custom visual editors. These editors serve for editing each of the three following models: *Classes Diagram*, *State Machine Diagram* and *Objects Diagram*. Obtained diagrams represent a part of a project. Source code for this part is generated automatically on *C#* programming language [7].

### 3.1. Classes Diagram

*Classes Diagram* is intended for visual creation of *classes*, *interfaces* of the project and setting of its relationships. *IDE Visual Studio 2005* has its own classes diagram which also allows creating classes, interfaces and so on. It has some disadvantages though: it is not always synchronized with a source code. For example, if we create any class on standard class diagram and then delete this diagram created class will stay in the project. And wise versa if we create a class independently of the diagram this class will appear on the diagram only after diagram's regeneration and saving. It is unacceptable to have a source code and diagrams which are different. Diagram and source code should always be synchronized to prevent misleading situations.

*Classes Diagram* introduced in this paper is completely synchronized with the code it reflects. This diagram also has possibilities for adding of automata behaviour for the classes on it. On adding automata behaviour from *Classes Diagram* to some class automatic transition to the *State Machine Diagram* is performed.

*Classes Diagram* also has validation mechanism which prevents creation of wrong constructions in meaning of C# language concepts.

### 3.2. State Machine Diagram

*State Machine Diagram* is intended for creation of visual models for the objects which lifecycle could be represented by finite count of states with guarded transitions between them.

*State Machine Diagram* has validation mechanism which allows guaranteeing the following rules:

*1.* State machine has only one initial state.

*2.* State machine has at least one final state.

*3.* Every transition has specified event on which this transition is performed.

*4.* Every state should be reachable from the initial state.

*5.* Transitions with the same event and same source state should have orthogonal guard conditions.

The fifth rule is very hard and has no accurate solution in this work.

### 3.3. Objects Diagram

*Object Diagram* is intended for visual creation of starting configuration of the application. For each object on the diagram instance of specified type is created and developer are able to set initial values of properties for each object, set relationships between different objects. And if there is object with automata behaviour on the diagram, developer can mark corresponding state machine as a start point of the whole application.

### 3.4. Code generation text templates

Code generation for specified model is possible with *DSL Tools*. Code generation is performed using text template transformation. Text templates are different for each of the three diagrams. On every model change new source code is generated. Thanks to this approach source code always completely represent visual model for any of the three diagrams.

Developer should never modify auto generated source code because it will be overwritten on the next saving or compilation of the project.

## 4. Usage

*State Machine Designer* could be used during development of a new project in *Visual Studio 2005* and during adding of new functionality to existing project as well. Usage of *State Machine Designer* is useful in almost all cases when developer wants to add some automata behavior to a project.

When *State Machine Designer* plug-in is installed developer can add new entities in his project and edit diagrams using implemented editors. There are three items which are corresponding to diagrams: *ClassesLanguage*, *StateMachineLanguage* and ObjectsLanguage. Files with those extensions automatically use custom editors which are described below.

## 4.1. Classes Diagram editor

For editing of *Classes Diagram* developer adds new item to the project with special extension *ClassesLanguage*. When developer double-clicks on such an item special editor is appeared in new document window (Figure 1).

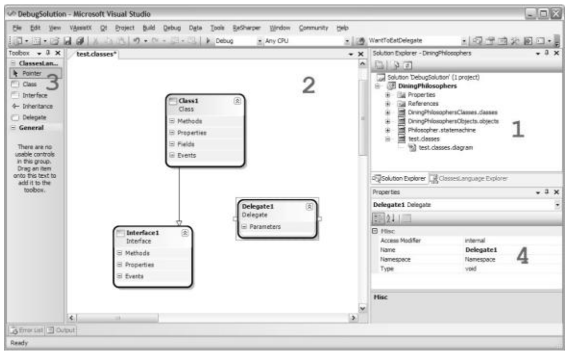


Figure 1. Classes Diagram editing in Visual Studio 2005.

Digits on the picture specify the main windows of *IDE* during classes editing:

1. *Solution Explorer* – tree list of modules and files in the project. Using *"Add New Item …"* command developer can add new files to the project as well as files with *ClassesLanguage*, *StateMachineLanguage* and *ObjectsLanguage* extensions which represent supported by tool diagrams.

2. *Classes Diagram* editor area. Developer is able to add classes and interfaces to this area using extended toolbox.

3. Toolbox which contains items for adding classes, interfaces and inheritance relationships to *Classes Diagram*.

4. Property window for the diagram‘s active object.

Using this diagram adding of automata behaviour to any class is possible. After right click on any class figure on the diagram context menu is shown (Figure 2). Developer can choose *"Add/Edit automat behaviour"* item and active window will be automatically switched to the *State Machine Diagram* for the chosen class. If there is no such a diagram it will be automatically created.

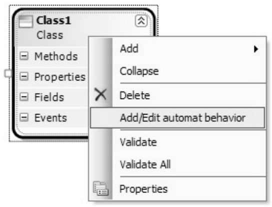


Figure 2. Context menu for the class on Classes Diagram.

All classes which have its own state machine are drawn with "A" icon in the top-left corner (Figure 3).

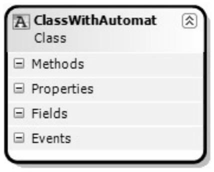
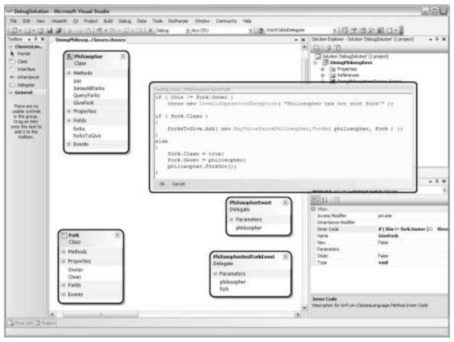


Figure 3. Class with automata behaviour.

Implementation for all methods for all classes from the diagram could be done using additional code editor which is shown after double click on any method (Figure 4).



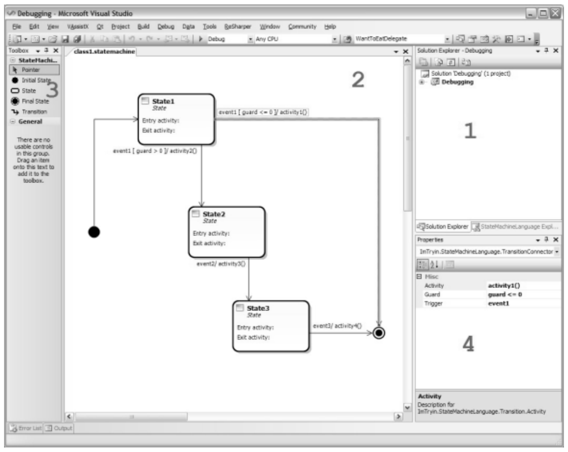
```
if ( this != fork.Owner )
    throw new InvalidOperationException( "Philosopher has not such fork!" );

if ( fork.Clean )
{
    forksToGive.Add( new KeyValuePair<Philosopher,Fork>( philosopher, fork ) );
}
else
{
    fork.Clean = true;
    fork.Owner = philosopher;
    philosopher.ForkGot();
}
```

Figure 4. Additional source code editor.

*Classes Diagram* supports model validation. For example, it doesn't allow inheritance of interface from the class or class from two or more classes because such constructions are wrong in .NET languages. If some of the restrictions are not satisfied message with corresponding error is shown in error window.

### 4.2. State Machine Diagram editor

For editing of State Machine Diagram developer adds new item to the project with special extension *StateMachineLanguage*. Such items also have their own editor (Figure 5). *StateMachineLanguage* item could be created automatically when developer adds automata behavior to some class on the classes diagram.


Debugging - Microsoft Visual Studio
Toolbox
StateMachi...
Pointer
Initial State
State
Final State
Transition
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
class1.statemachine
State1
State
Entry activity:
Exit activity:
event1 [ guard <= 0 ]/ activity1()
event1 [ guard > 0 ]/ activity2()
State2
State
event2/ activity3()
State3
State
event3/ activity4()
Solution Explorer - Debugging
Solution 'Debugging' (1 project)
Debugging
Solution Explorer
StateMachineLanguage Expl...
Properties
1
2
3
4

InTryin.StateMachineLanguage.TransitionConnector

Misc

| | |
|---|---|
| Activity | activity1() |
| Guard | guard <= 0 |
| Trigger | event1 |

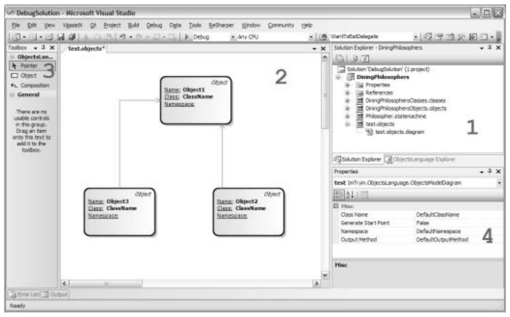
Activity
Description for InTryin.StateMachineLanguage.Transition.Activity

Figure 5. State Machine Diagram editing in Visual Studio 2005.

Digits on the picture mean the same areas as on Figure 1. Toolbox for state machine editor contains *"Initial state"*, *"State"*, *"Final state"* and *"Transition"* items. Using this items developer visually creates state machine. *State Machine Diagram* also supports validation. The following rules are always satisfied: there is only one initial state, there are no unreachable states and so on.

### 4.3. Objects Diagram editor

For editing of *Objects Diagram* developer adds new item to the project with special extension *ObjectsLanguage* (Figure 6).



test InTryin.ObjectsLanguage.ObjectsModelDiagram

Misc

| | |
|---|---|
| Class Name | DefaultClassName |
| Generate Start Point | False |
| Namespace | DefaultNamespace |
| Output Method | DefaultOutputMethod |

Misc

Figure 6. Objects Diagram editing in Visual Studio 2005.

This diagram is useful when application has fixed count of different objects. In this case objects configuration could be represented on the diagram. And those objects which have automata behaviour will be having "A" icon in the top-left corner of the shape (Figure 7).

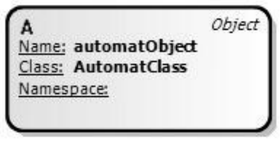


Figure 7. Object with automata behaviour.

On *Objects Diagram d*eveloper could assign some of state machines to start when the whole application is started.

## 5. Conclusions

This paper describes a tool for *Microsoft Visual Studio 2005* which allows developer to model and develop application using three diagrams: *Classes Diagram*, *State Machine Diagram* and *Objects Diagram*. This tool extends abilities of *Microsoft Visual Studio 2005* in applications designing and development. With this approach a part of the source code is generated automatically that follows to decreasing of errors count. Thanks to visual models application becomes more clear and logical.

Developed tool could be used in development of any application but it is most helpful in cases of reactive systems [8]. Usage of the tool in development helps programmer to see static and dynamic models of the application simultaneously that also makes development process easier.

Currently there is no similar tool for *Visual Studio 2005* which will help developer to combine traditional programming techniques with automata-based programming.

## 6. References

[1] Dan Pilone, Neil Pitman, *UML 2.0 in a Nutshell*, O'Reilly, 2005.

[2] Tukkel N.I., Shalyto A.A., "State-based programming", *PC World*, 2001, #8, pp.116-121; #9, pp.132-138.

[3] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns*, MA: Addison-Wesley Proffesional, 2001, p.395.

[4] Shalyto A.A., "SWITCH-Technology. Algorithmization and Programming of Logic Control Problems", St. Petersburg: Nauka, 1998.

[5] Microsoft Corporation, *Microsoft Visual Studio 2005*, http://msdn.microsoft.com/vstudio/.

[6] Microsoft Visual Studio Developer Center, *Domain-Specific Language Tools*, http://msdn.microsoft.com/vstudio/DSLTools/.

[7] Jesse Liberty, Brian MacDonald, *Learning C# 2005*, O'Reilly, 2006.

[8] Harel D. et al. "Statemate: A working environment for the development of complex reactive systems", *IEEE Software Eng.*, 1990, #4.

# Application of Automata-Based Programming for Construction of Business Processes Management Systems

Evgeny Andreevich Mandrikov Vladimir Anatolievich Kulev

***Abstract*—In this article the problem of business processes modeling languages consolidation in a uniform management system is considered. It will be shown that it is reasonable to use automata-based programs as a base for proposed system.**

## I. Introduction

The main objective of business process management systems is the automation of the processes taking place in business [1]. When building software products, working with business processes, there are three basic roles: the end-user, business analyst and developer. Business analyst examines and describes the business process and formulates the requirements for software, the developer implements them in the final product.

Traditional business process management systems are trying to be based on the model created by business analyst, and build executable program on the basis of this. Such a way of automating business processes, has gradually forced to move away to second place due to the fact that making even small changes in the business process logic means time-consuming and expensive reprogramming. As a result, applications are not able to be updated at the desired speed, dictated by changing business conditions and needs of the enterprise [2]. It should also be noted that software products developed in such manner are deeply attached to the specific business processes and end-users. This does not allow software to be rapidly adopted in other enterprises.

There is a trend of replacing traditional way of automating business processes with usage of various domain-specific languages [3], based on the description of specific types of business processes, such as: task management (Issue Tracking, Bug Tracking) [4], document management (Document Management System) [5], etc. Using this approach, business analyst and the developer communicate in one language with a graphical representation of the process. Business analyst is responsible for the graphical representation and should not deal with technical details of the process. But without these details the business process is not fully defined and thus can not be done, so the developer is responsible for their program implementation.

Most of existing software products allow usage of only one or few languages to describe business processes. This makes it difficult to integrate various business processes that occur on the same enterprise, and force developers to do more work. This article describes the approach to solving this problem by translating business processes descriptions into automata programs [6].

E. A. Mandrikov is MSc student at the Computer Technologies Department, Saint-Petersburg State University of Information Technologies, Mechanics and Optics, Saint-Petersburg, Russia (e-mail: mandrikov@rain.ifmo.ru).

V. A. Kulev is MSc student at the Computer Technologies Department, Saint-Petersburg State University of Information Technologies, Mechanics and Optics, Saint-Petersburg, Russia (e-mail: kulev@rain.ifmo.ru).

The research is supervised by A. A. Shalyto, PhD, professor at the Computer Technologies Department, Saint-Petersburg State University of Information Technologies, Mechanics and Optics, Saint-Petersburg, Russia (e-mail: shalyto@mail.ifmo.ru).

## II. Business processes description languages

BPEL[1] – a widely used language for describing business processes, designed to work in the web-services environment [7]. It is based on WSDL[2], and while WSDL allow the use of JavaBeans, the natural choice is web-services.

The deployment of business process in BPEL leads to the publication of a web-service, which is the primary interface for interaction with the process. Variables within the BPEL are XML fragments or XSD[3] basic types. BPEL hac structures to describe the control flow logic and call other WSDL services. Ultimately, BPEL – a language for describing web-services management business processes.

Another popular language is jPDL, created by the Jboss jBPM project [8]. One of its main purposes is task management. For this the language has special design to allow creating tasks in process. Changes in tasks (such as beginning or end of execution) are also events which business process is driven by. Also jPDL supports asynchronous execution of tasks and business process branching into several parallel threads.

By the example of BPEL and jBPM it can be seen that the existence of domain-specific languages is feasible because of broad spectrum of challenges faced in enterprise automation. But it is also easy to see that these languages have many common features, in particular the representation model of business process in the form of a graph.

[1]Business Process Execution Language
[2]Web Services Description Language
[3]XML Schema Definition

## III. Translation of business processes descriptions into automata programs

The proposed approach is to translate business processes descriptions into a system of interacting state machines. They may interact:

- by nesting – a machine nested into one or more states of the other machine;
- by calls – a machine is called on a certain event generated by some transition;
- by messages exchange – a machine receives a message from another one;
- by states – a machine checks the state of another machine.

Figure 1 is an example of the simplest process description in jPDL. This example will show how business process can be translated into automata program.

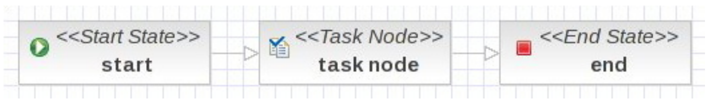


Figure 1. An example of business process description in jPDL

In this example, task node is translated into nested state machine, managing the internal state of the task. This automata program is shown on Figure 2.

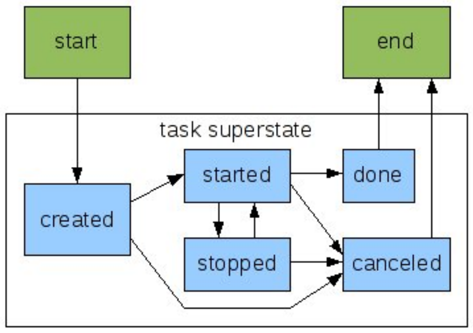


Figure 2. An example of resulting automata program

Similarly, all other high-level business processes languages can be translated into automata programs. Resulting programs can be executed in a single environment for all processes (Process Virtual Machine), or translated into executable code.

## IV. Conclusion

The benefits of the proposed approach are:

- possibility of a business analyst using conventional business processes modeling software;
- ease of integration of different business processes description languages in a single system;
- extensibility of the system due to possibility of adding new languages;
- possibility of business processes verification (in fact, verification of resulting state machines [9]).

## References

[1] “Business process management.” [Online]. Available: http://en.wikipedia.org/wiki/Business_process_management

[2] V. Kleban and F. Novikov, “Application of finite state machines in the document flow,” *Scientific and technical bulletin of SPbSU ITMO*, no. 53, Automata programming, pp. 286–294, 2008.

[3] “Domain-specific programming language.” [Online]. Available: http://en.wikipedia.org/wiki/Domain-specific_programming_language

[4] “Issue tracking system.” [Online]. Available: http://en.wikipedia.org/wiki/Issue_tracking_system

[5] “Document management system.” [Online]. Available: http://en.wikipedia.org/wiki/Document_management_system

[6] “Automata programming homepage,” Programming Technologies dpt., SPbSU ITMO. [Online]. Available: http://is.ifmo.ru/

[7] “Business process execution language.” [Online]. Available: http://en.wikipedia.org/wiki/BPEL

[8] “Jboss jBPM.” [Online]. Available: http://www.jboss.com/products/jbpm/

[9] E. Kurbatsky and A. A. Shalyto, “Verification of programs built on automata-based approach,” *Scientific software in education and scientific research. SPbSPU.*, pp. 293–296, 2008.

# Declarative Language for SAX Handler Definition

Alexey Vladykin
*St. Petersburg State University of Information Technologies, Mechanics and Optics*
*vladykin@gmail.com*

## Abstract

*In this paper a declarative language for SAX handler definition is proposed. This language allows to describe complex XML parsing algorithms in a simple manner. An algorithm is introduced for automatic transformation of such handler descriptions into finite state machines, and then into source code. This approach reduces the complexity of SAX handler development by eliminating the greater part of error-prone manual work.*

## 1. Introduction

XML is widely used for a variety of purposes: from storing program configuration to transmitting data packets over the Internet. More and more applications require XML support, and programmers often have to develop code that extracts data from XML documents.

Some XML documents are very large: up to hundreds of gigabytes. E. g. a complete Wikipedia dump including all articles with their change history takes 148 Gb (bzip2-compressed).

How can one parse such document and extract some useful information from it? The only feasible approach for documents like Wikipedia dump is Simple API for XML (SAX) [3], because SAX parser is very effective about computer resources and passes XML content to the rest of application in small portions. Other well-known approaches to XML parsing like Document Object Model (DOM) [6] and Java API for XML Binding (JAXB) [1] need to load the whole document into RAM, which is unacceptable.

The major SAX drawback is complexity of crafting SAX handlers manually. [4] This paper describes an approach which significantly simplifies development of SAX handlers. A declarative language for XML handler definition is introduced. Handler definition in this special language is automatically translated into finite state machine, and then into source code in any programming language.

## 2. Simple API for XML

SAX is a low-level XML parsing technique. SAX parser sequentially reads input document and notifies SAX handler about every start and end tag, as well as about character data between tags.

SAX parser implementations exist for many programming languages as part of standard library or as a 3rd party library.

SAX handler must be written by hand for each document type that needs to be processed. Crafting SAX handlers may be very hard task in case of complex document structure.

Handler class in Java extends class *DefaultHandler* and typically overrides the following three methods with hand-written code:

*void startElement(String uri, String localName, String qName, Attributes attrs)*

*void endElement(String uri, String localName, String qName)*

*void characters(char[] ch, int start, int length)*

Handler calls *startElement* method when it encounters start tag, *endElement* is called for end tag, and *characters* – for character data between tags. [2]

For other languages SAX handler structure is essentially the same.

## 3. SAX Handler as an Entity with Complex Behavior

It is important to understand the reasons why writing SAX handlers is so complex, and to eliminate those reasons. This task can be accomplished with automata based approach. [7]

SAX handler receives notifications from SAX parser and translates those notifications into commands or data structures that can be understood by the rest of the application. It has to track current parser position within the document and, according to that position, handle notifications differently.

SAX handler is an entity with complex behavior [7], because its reaction to incoming notifications depends on previously received notifications. Handler can be logically divided in two parts:

a) **controlling part**, which tracks current parser position in XML document by remembering the history of incoming notifications, and chooses one of the possible commands for controlled part;

b) **controlled part**, which receives commands from controlling part and translates them into commands or data structures for the rest of the application.

The biggest challenge usually is controlling (behavioral) part, because tracking of current parser position in XML document within traditional approach requires setting and checking many boolean flags, or other tricks. Controlled part is typically simple, but its code is spread and lost inside controlling part.

According to the paradigm of automata based programming, an entity with complex behavior should be represented as an automated object. Explicit separation of controlling and controlled parts of SAX handler can help crafting such handlers and make their structure and behavior much clearer. Moreover, it is possible to generate the code of controlling part based on a declarative definition. The next section describes a language that can be used for declarative definition of SAX handlers.

## 4. Declarative Language for SAX Handler

We'll demonstrate the language and its usage on a simple problem of extracting a list of departments and all non-terminated employees from an XML document describing company structure.

Such document could look like this:

```
<company>
<name>Mr. X and Partners</name>
<department>
   <name>Human Resources</name>
   <employee>
      <name>John Smith</name>
   </employee>
   <!-- more employees -->
</department>
<department>
   <name>Engineering</name>
   <employee terminated=”true”>
      <name>Zzyzzy Zzyrryxxy</name>
   </employee>
   <!-- more employees -->
</department>
<!-- more departments -->
</company>
```

Document structure shown here is relatively simple, but it exposes a typical problem of distinguishing between *name* of a company, *name* of department and *name* of employee.

Another typical problem here is that extracting *John Smith* from fragment <*name*>*John Smith*</*name*> requires adding some lines of code to all three of SAX handler methods: *startElement*, *endElement* and *characters*. Thus a logically atomic extraction of employee name is split into several stages.

Both mentioned problems are related to the controlling part of SAX handler and imply saving handler state between invocations of its methods. This is what makes writing SAX handlers complicated.

Let's see how these problems can be solved when SAX handler is described using the proposed declarative language. Handler definition in the proposed language looks like:

```
(
<department>
   <name> { capture(); }
   </name> { obj.addDepartment(captured()); }
   (
   <employee terminated!=”true”>
      <name> { capture(); }
      </name> { obj.addEmployee(captured()); }
   </employee>
   )*
</department>
)*
```

This definition consists of the following elements:

a) start and end tags. Start tag may have constraints on its attributes (e.g. *terminated!=”true”*). Arbitrary code may be specified after tag in braces. It will be executed when such tag is encountered in document.

b) parentheses, used to group tags and specify zero-or-one (?) and zero-or-more (*) quantifiers, or enumerate alternatives separated by |.

Here is the formal grammar for this language:

```
S :: START_TAG | END_TAG | GROUP
START_TAG :: "<" ID OR_EXPR? ">" ACTION?
END_TAG :: "</" ID ">" ACTION?
GROUP :: "(" ( START_TAG | END_TAG
      | GROUP )* ( "*" | "?" )? ")"
ACTION :: "{" CODE "}"
OR_EXPR :: AND_EXPR ( "||" AND_EXPR )*
AND_EXPR :: TERM ( "&&" TERM )*
TERM :: ID "==" STRING | ID "!=" STRING
      | ID "=~" STRING | ID "!~" STRING
      | "(" OR_EXPR ")"
```

In this grammar *ID* is any valid tag or attribute name; *STRING* is arbitrary text enclosed in quotation marks or keyword *null*; *CODE* is arbitrary code in target programming language.

According to the paradigm of automata based programming our SAX handler will consist of two parts: controlling part (automaton) and controlled part. Declarative definition of controlling part is shown above.

Controlled part provides some interface to controlling part. In our case this interface consists of two methods: *void addDepartment(String name)* and *void addEmployee(String name)*. Implementation of these methods and the whole controlled part class is up to the developer.

Controlling automaton calls methods of controlled object according to the declarative definition shown above. Additionally it can use two utility methods provided by controlling automaton: *void capture()* and *String captured()*. The former tells automaton to capture character data coming from parser into temporary buffer. The latter returns character data captured since last call to *capture* and clears the buffer.

## 5. Building Automaton for SAX Handler

To build controlling automaton from its declarative definition we need to extract its states and build transitions between states.

Both tasks are rather simple, because states and transitions are implicitly present in the declarative definition. Informally speaking, each place between tags corresponds to one state, and tags correspond to transitions.

There are some corner cases related to parentheses and quantifiers, but overall algorithm is pretty straightforward. Its implementation in Java can be found at project website [5].

## 6. Code generation

Given the description of SAX handler controlling automaton as a set of states and transitions, it is possible to automatically generate source code in virtually any programming language. Currently implemented is code generation for Java. [5]

Code generator creates a class that extends *DefaultHandler* and overrides aforementioned methods *startElement*, *endElement* and *characters*. All transitions on start tags are placed in *startElement*, and all transitions on end tags – in *endElement*.

Automaton has only two member variables: current state (integer) and temporary buffer for character data. Thus memory consumption is minimal.

Every call to automaton's *startElement*, *endElement* or *characters* is a quick constant time operation (not taking into account what happens in invoked methods of controlled object), so parsing XML document with automated SAX handler remains an efficient linear algorithm.

## 7. Conclusion

In this paper a declarative language for SAX handler definition is introduced, and an automatic code generation system is described. The system takes declarative definition of SAX handler as input, builds controlling automaton and then translates it into source code in some programming language. Currently code generation is implemented for Java programming language. Implementation of the remaining part of SAX handler – controlled object – is to be written manually by the programmer, but this part is typically trivial. Proposed approach features significant simplification of SAX handler creation, and does not bring any performance penalty.

This approach has been used in development of an application that extracted contents of several kinds of complex table-like structures from thousands of XML documents.

## 8. References

[1] JAXB Reference Implementation. https://jaxb.dev.java.net/

[2] McLaughlin B. *Java and XML, Second Edition*. O'Reilly, 2001.

[3] Official website for SAX. http://www.saxproject.org/

[4] Oleg Kiselyov. “A better XML parser through functional programming”. *LCNS*, Springer-Verlag, 2002, pp. 209-224.

[5] SaxGen – Google Code. http://code.google.com/p/saxgen/

[6] W3C Document Object Model. http://www.w3.org/DOM/

[7] Поликарпова Н. И., Шалыто А. А. *Автоматное программирование*, Питер, СПб., 2009.

# The Formal Approach to Computer Game Rule Development Automation

Elena A. Pavlova

***Abstract*—Computer game rules development is one of the weakly automated tasks in game development. This paper gives an overview of the ongoing research project which deals with automation of rules development for turn-based strategy computer games. Rules are the basic elements of these games. This paper proposes a new approach to automation including visual formal rules model creation, model verification and model-based code generation.**

***Index Terms*—Automation, Games, Formal Languages, Software Verification and Validation**

## I. Introduction

COMPUTER games are one of the most dynamic and rapidly evolving fields of information technology. Games are widely used in entertainment, education and training of personnel [1]. Still the percentage of successful projects in computer game industry is very low [2]. Low level of automation is one of the reasons of the problem. Game rule development is one of the weakly automated tasks. Game rule development includes rule design, rule-based code and data generation and results verification [3].

In this paper we define game rules as the definition of a game world entities, entity interaction rules, the main goal of the game, secondary goals, start conditions, winning conditions and a player state definition.

There is a special role of game designer in a game development team [3, 4], who is responsible for game rules design. She frequently has no technical background. In order to avoid confusion between a game designer and a software designer roles we'll use hereinafter the term "designer" for a game designer.

A game rules definition usually consists of several large text documents and a set of tables. Designers use text editing tools and spreadsheets as automation tools. The typical process of a game rules definition is shown in Figure 1

Manuscript received March 14, 2008. E. A. Pavlova is PhD student at the Cybernetics Department, Moscow Engineering-Physics Institute (National Nuclear Research University), Moscow, Russia (phone: +7-906-723-1189; e-mail: elena.pav@gmail.com).

The research is supervised by M. V. Sergievsky, PhD, associate professor at the Cybernetics Department, Moscow Engineering-Physics Institute (National Nuclear Research University), Moscow, Russia (e-mail: maxim.sergievsky@light-site.ru).

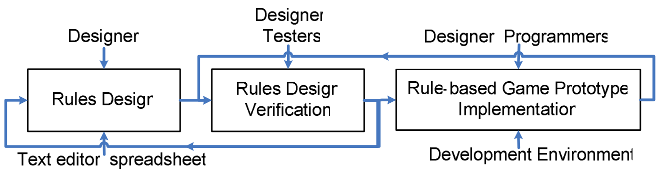


Fig.1. Computer game rule development process.

Voluminous (up to several thousand pages [3]) and ill-structured rules definitions are difficult and time-consuming to create, modify and maintain. An informal rules formulation complicates rules development automation, automatic rule-based code and data generation and rules verification, particularly rules balance checking and balancing (balance problem will be discussed further).

Some designers try to master general-purpose modeling and programming languages and corresponding modeling environments to solve the automation problem (this approach is described in detail in [3, 4]).

Special-purpose game-oriented development environments are used as an alternative (see for example [5-9], Torque Engine Advanced (www.garagegames.com/products/torque/tgea/features/), FPS Creator (www.darkgamestudio.com), NeoAxis Engine (www.neoaxisgroup.com), Offset Engine (www.projectoffset.com/game.html), Unreal Engine (www.unrealtechnology.com), C4 Engine (www.terathon.com/c4engine)). These tools frequently need a complex customization and add-in programming. The majority of tools don't allow for game genre-specific development, thus loosing long reusable experience. The rules definition is mixed with the definition of the graphics of the game, and in some cases with the definition of game artificial intelligence elements. Thus, it is difficult to modify, analyze, verify and reuse game rules independently. The rapid game rules executable prototype generation becomes almost impossible.

The tools under discussion don't allow automatic rules verification. Design-time errors could be caught at the implementation stage or later. These errors are usually treated by a manual rule definition documents review and tedious testing [3, 4].

The above demonstrates the urgency of development of the new approach to game rules automation. The goal of this paper is to develop an approach allowing automation of the whole game rules development cycle – from formal visual rule design, through rule verification, to rule-based code and data generation. The approach should consider the domain-specific experience of the particular game genre. We took the turn-based strategy (TBS) game genre [3] for our research as game rules are the critical part for a game of this genre. The key task in TBS game development is game rules development [4].

As far as the author of this paper is concerned the creation of the rule development environment supporting visual game rule representation as a single formal model, automatic formal verification of this model and model-based automatic code and data generation is a new approach to game rules development automation. Formal domain-specific languages weren't used for TBS game rules definition before. Static analysis and formal property monitoring weren't used for computer game rules verification.

## II. Formal Background

### A. The Turn-Based Strategy Game Rule Description Language

The developed language is domain-specific (DSL) [10], i.e. it considers a specificity of turn-based strategy computer games. The language allows entity definition for the problem domain of the concrete game (notably entity data and behavior); behavior constraints; entity relationships and reactions of entities to the other entity behavior.

Entities are represented as objects in the language. Entity data correspond to object properties, and behaviors correspond to methods. Object orientation allowed more flexibility and simplicity compared to class orientation. The language type system was developed. The language type system includes simple types for evaluation of constraints and game genre-specific types (types for the turn-based strategy domain). The type system provided the necessary level of abstraction and allowed to separate the object specification from the implementation. Types are also used for model error detection.

The language is prototype-based [12]. The new (clone) object maintains the independent copy of properties, methods and the link to the initial (prototype) object. An object may have only one prototype object. The modification of prototype object doesn't influence the clone object and vice versa. The main object modification method is property and method update. The specified prototyping mechanism allowed object elements reuse and seems to be the natural object-creation mechanism for the turn-based strategy games domain.

The language syntax was formally defined using a context-free LL(1) grammar [13]. Both textual and graphical notations for the language are available. We defined the formal denotational semantics [13] for the language, using the $\lambda\varsigma$ – calculus [12] for denotats and elements of Hoare logic [13] for precondition and postcondition behavior constraints. Conditions described in Hoare logic are used for model consistency checking and correct method invocation planning in game scenarios not for verification by deduction analysis. Entities react to the behavior (i.e. method invocation) of other entities by means of the special methods called reactors. Several reactors may be attached to one method. Reactor execution changes the state of the game. Reactors' preconditions and postconditions depend on the game state. Thus the reactors invocation order is important. Reactors' invocation algorithm considers reactors' preconditions and postconditions to execute as much reactors as possible. The developed language is described in detail in [11].

### B. Turn-Based Strategy Game Rule Verification

Design-time game rule verification allows incorrect rule detection and following correction prior to implementation. This paper defines a correct rules model as consistent and balanced. The rules model consistency is defined as syntactic and semantic interface consistency of objects constituting the model. Model consistency guarantees the correct interaction of objects. The applied consistency checking method is fully described in [14].

Balance is a game domain-specific concept. In general it means rules fairness [3]. The rules of a particular game are fair if the player success depends only on his abilities. Concrete definitions of balance are given in [3, 4]. The example of a misbalanced game is a game having unequal start conditions for players, giving one player the advantage allowing winning no matter what other players do. Rules balance verification is a key task in TBS game development [3]. In this paper rules are considered to be balanced if none of the competing sides defined by the rules has an advantage; there are no invincible troops and the result of the game is independent of who moved first. In this paper rules balance is verified by means of formal properties monitoring [15].

## III. The Prototype Rules Development Environment

The architecture of the designed rule development environment is considered in this section. The architecture model is illustrated in Figure 2. The arrows connecting model elements represent dataflows. The rules development environment consists of the graphical user interface allowing visual game rules model creation, the rules verification tool, the rules translator and the data (rules models) storage.

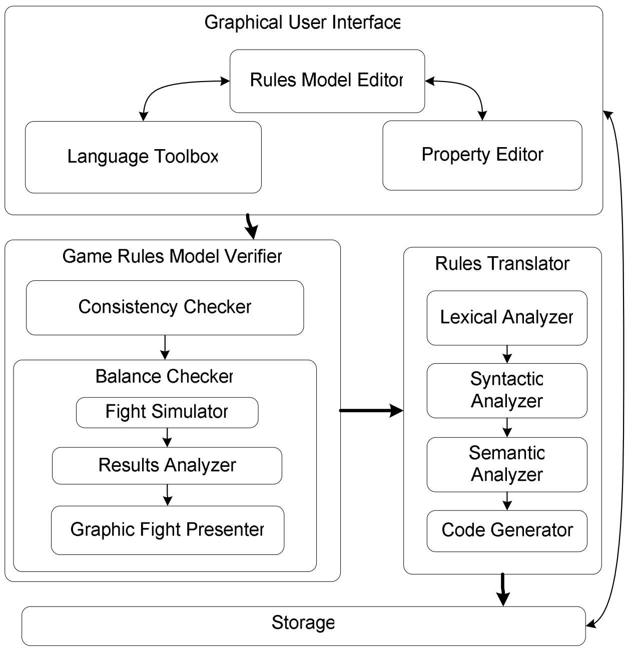


Fig. 2. The rules development environment architecture.

The graphical user interface enables visual rules definition in the rules model editor. Rules are defined using the graphical language notation. The example screenshot of the prototype rules development environment is shown in Figure 3.

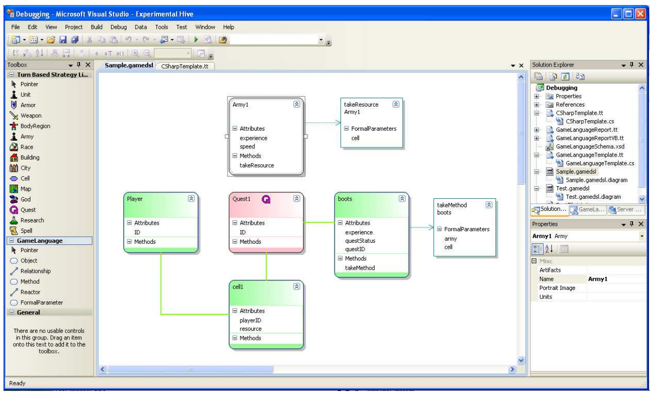


Fig. 3. The rules development environment and the example rule model.

The toolboxes contain all necessary language graphical primitives. Model element properties could be customized in the property editor. The model description is stored in the textual language notation.

The verification tool checks for rules model consistency and balance using methods defined in the section I.B. The translator transforms the verified textual rules representation into the C++ code. This programming language is considered to be the most popular for game development.

We developed the prototype rules development environment for turn-based strategy game rule development support. The prototype was developed using Microsoft Visual Studio 2008 SDK that enabled using managed code and rapid language and visual modeling tool creation.

## IV. Related Work

There are several approaches reported in literature for dealing with the game rules development automation problem. Moreno-Ger *et al.* [5] consider adventure games creation for educational purposes. The textual adventure game-specific language and the corresponding interpreter are created. One will need to master the textual notation of the language to use the proposed environment. That might complicate a game designer's job. Rule verification is not considered. Moreno-Ger *et al.* [6] extend the environment proposed in [5] by new reusable adventure game-specific entities.

Hu [7] also considers adventure games for education. The proposed education model describes education roles (a teacher, a student, a course-book, etc.). The education model-based TorqueEngine script extension is developed. The rules are stored in several files using basic TorqueEngine principles. Rule verification is not considered. The educational adventure game development is significantly restricted by the education model proposed.

Furtado *et al.* [8] suggest an approach and a general framework for game development. The informal visual modeling language is defined and the corresponding tool is created. The language is intended to describe game rules, game graphics and sound. The language operates the notions of "game state", "program", "audio component", etc.. Two layers of abstraction are mixed in a single game model (the game components layer and the specific component object model layer). That may complicate game designer's job. The proposed framework supports the model inspection for states reachability, states existence and constraints existence. Balance checking is not supported.

Amory [9] deals with educational quest and adventure games development automation. The approach includes the development of interface library encapsulating concepts from the corresponding game genre domains. An educational quest or an adventure game could be created implementing the interfaces of this library. Rules verification is not considered.

Thus, the key advantages of our approach is separation of the rules and the graphics definitions, the design-time rules verification possibility (including balance verification) and turn-based strategy game genre specificity consideration.

## V. Conclusions and Future Work

In the previous sections we presented a detailed description of the new approach to game rule development automation for the turn-based strategy game genre. The approach includes the development of a visual formal rule description language, a formal rule verification method and a rules development environment supporting single formal rule model creation, verification and rule-based executable prototype generation. Following this approach we developed the necessary formal basis and the prototype tool for game rules development. The prototype considers turn-based strategy game-specific experience, allows rule balance verification, rapid rules prototype development and rule reuse.

The developed language simplifies rules development. The application of domain-specific languages to TBS rules definition is a new approach to TBS development. The application of formal verification at design-time allowed error detection prior to implementation.

The main advantages of the proposed approach are as follows: rules definition and game graphics definition separation, rules verification automation and TBS genre-specific knowledge consideration.

Additional approach improvements include further development of the balance verification method. We are going to broaden the definition of balance and check for the so-called dynamic balance [3], i.e. the balance at every turn of the game. So players could be guaranteed positive experience throughout the game. We plan the extension of the language type system and the object cloning mechanism revision. Finally, it is planned to extend the approach to the real-time strategy game genre which is very similar to the turn-based strategy genre. This will introduce a concept of mission (missions do not exist in turn-based strategies) and will lead to complex time-constraints consideration. It is expected that our new results will facilitate the development of a development environment for strategy games creation.

## REFERENCES

[1] M. J. Taylor, M. Baskett, G. D. Hughes, S. J. Wade, “Using soft systems methodology for computer game design,” *Systems Research and Behavioral Science,* #24., pp. 359-368, 2007.

[2] M. Brydon, A. Gemino, “Classification trees and decision-analytic feedforward control: a case study from the video game industry,” *Data Mining and Knowledge Discovery,* vol. 17 , Issue 2, pp. 317–342, 2008

[3] A. Rollings, D. Morris, *Game Architecture and Design. A New Edition*. Indianapolis: New Riders Publishing, 2004.

[4] G. Wihlidal, *Game Engine Toolset Development.* Boston, MA: Thomson Course Technology PTR, 2006.

[5] P. Moreno-Ger, I. Martinez-Ortiz, J. L. Sierra, B. Fernandez-Manjon, “Language-driven development of videogames: the <e-Game> experience,” *Entertainment Computing - ICEC 2006*, pp. 153-164, 2006.

[6] P. Moreno-Ger, J. L Sierra., I. Martinez-Ortiz, B. Fernandez-Manjon, “A documental approach to adventure game development,” *Science of Computer Programming*, vol. 67 , Issue 1, pp. 3-31, Jun. 2007.

[7] W. Hu, “A reusable eduventure game framework,” *Transactions on Edutainment I* , pp. 74-85, 2008.

[8] A. W. B. Furtado, A. L. M. Santos, G. L. Ramalho, “A computer games software factory and edutainment platform for Microsoft .NET,” *IET Software*, vol. 1, Issue 6, pp. 280 – 293, Dec. 2007.

[9] A. Amory, “Game object model version II: a theoretical framework for educational game development,” *Educational Technology Research and Development*, vol. 55, Number 1, pp. 51 – 77, Feb. 2007.

[10] J. Greenfield, K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools,* 2004.

[11] E. A. Pavlova, “Design of formal domain-specific language for computer game rules development for turn-based strategy game genre,” *Computer and Information Technology Reporter*, Feb. 2009. (in Russian).

[12] M. Abadi, L. Cardelli, *A Theory of Objects*, 1996.

[13] P. D. Mosses (2002). Fundamental Concepts and Formal Semantics of Programming Languages – An Introductory Course. [Online]. Available: http://wiki.daimi.au.dk/dSprogSem-01, 2002.

[14] A. V. Gavrilov, E. A. Pavlova, “Functional interface analysis for formalization of complex information system design,” *Information Technologies*, #9, pp. 9-15, 2008. (in Russian).

[15] V. V. Kulyamin, “Software verification methods,” *Russian State Analytical-Review Articles Contest for Priority Concept “Information-Telecommunication Systems*, 2008. (in Russian.).

# Information System of Scenario Strategic Planning

Denis R. Tenchurin
*State University - Higher School of Economics, Moscow, Russia*
*dtenchurin@gmail.com*

Maxim P. Shatilov
*State University - Higher School of Economics, Moscow, Russia*
*maxim.shatilov@gmail.com*

Scientific advisor: prof. Sergei M. Avdoshin
*State University - Higher School of Economics, Moscow, Russia*
*savdoshin@hse.ru*

## Abstract

*This paper gives an overview of the concept of a new system to support decision-making process in the area of strategic management company - DEM. To ensure that in the modern world the company remains leader in its industry it needs to continually adjust its development strategy to changes. Nevertheless, there is no unified methodological framework for strategic planning. Also there are no strategic planning systems to help managers make strategic decisions (a problem of choice of the one of the development strategies). Software Engineering Department Higher School of Economics, in collaboration with the EPAM company, conducted research in this area; the interim results have been examined in this work.*

## 1. Introduction

Today modern fast-changing market of goods and services is forcing companies to review their strategic decisions and develop new ones. In order to succeed companies should comprehensively evaluate the existing internal problems, and problems that may arise in the future. Moreover, they should be able to predict the impact of external factors on internal business processes, as well as the behavior of competitors and possible changes in the environment.

Based on the analysis of the strengths and weaknesses of contemporary strategic planning methodologies the new concept and its software prototype - Dynamic Enterprise Management (DEM) - were developed. The research was conducted within Software Engineering Department of Higher School of Economics in collaboration with EPAM Systems. DEM makes it possible to find the optimum of strategic development plan provided by the company. Thus with the right mix of resources the company can succeed.

As has been proved in practice, even a plenty of detailed information doesn't help managers to make, and the main thing, to implement strategic decisions. Thus the concept of hierarchical and balanced scorecard was proposed. This approach provides series of "instant snapshots" of the desired detail level that reflects the current situation in the company.

However, in order to actually organize a work process in the company, constantly evaluate the results, and adjust the development strategy of the company to changes, you need to answer some questions. How and where to take, or to measure the actual parameters, and how to relate assess their adequacy purposes with appropriate management processes, if necessary, to implement manages impact. In other words, a vicious management cycle with a feedback is needed (see Figure 1).

## 2. Balance Scorecard

The concept of Balanced Scorecard (BSC) was developed at the beginning of the 1990s by Robert Kaplan and David Norton from Harvard School of Economics. If the BSC was originally viewed as a tool for measuring and evaluating the effectiveness of the organization, in our time, it is a full-fledged methodology for strategic management.

Balanced Scorecard – is a description format of the organization for each of the four strategic aspects: internal business processes, development and training, customers, finance.

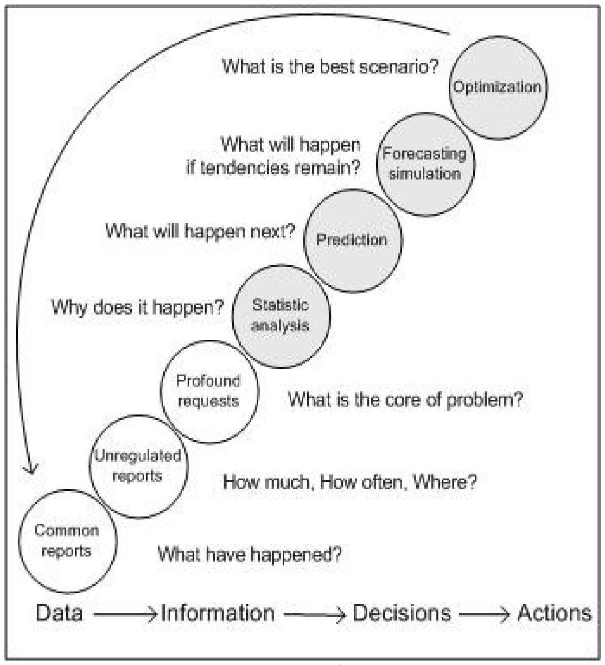


Figure 1

Each of the aspects contains main indicators for measuring the effectiveness of various processes. Another significant difference of BSC from its predecessors is the ability to organize treatment of intangible assets, all the more so in our time they play, of course, a great role [1].

Another advantage of the BSC is a special focus on the prospects of the company and their relationship with the current indicators. Correctly designed scorecard clearly shows a causal link between the current processes and prospects.

However, the BSC - is not a solution for all problems. Despite the fact that the system takes account of cause-and-effect relationships, it is very difficult to identify and build them on its basis. Moreover, a scorecard itself does not provide any assessments. Designing of the BSC cards - is a very laborious process, and adapting built maps to the processes of other companies is almost impossible.

However, BSC - is not a solution for all problems. Despite the fact that the system takes into account cause-and¬effect relationships, it is very difficult to identify and build them on its basis. Moreover, a scorecard itself does not provide any assessments. Designing of the BSC cards - is a very laborious process, and adapting built cards to the processes of other companies is almost impossible.

Despite of all the advantages of BSC, the methodology is more suitable for large scale simulation projects. If small and middle size projects are being developed it is more appropriate to use Goal-Question-Metric (GQM) framework [2]. In that case organizational goals are identified, then questions are developed to determine whether the goals are being met, and then metrics are identified that can help answer the arising questions. The framework was developed at the University of Maryland as a mechanism for formalizing the tasks of characterization, planning, construction, analysis, learning, and feedback. The GQM paradigm was developed for all types of studies, particularly studies concerned with improvement issues.GQM consists of three primary steps [3]:

1. Generate a set of organizational goals.
2. Derive a set of questions relating to the goals.
3. Develop a set of metrics needed to answer the questions.

The goals are built according to the needs of the organization, and they help in determining whether or not organization was improved the way it wanted to.

The comparison of BSC and GQM can be found in [4].

## 3. System Dynamics

Modeling - is one of the ways to solve problems virtually rather than by working with them in the real world. Modeling is used in cases where experiments with real objects are impossible or too expensive. It includes a display of real-world problems in the world of models and subsequent analysis, and optimization models, finding solutions, and display solutions back to the real world. The popularity of system dynamics can be explained be the following advantages of the methodology [5]:

- the majority of real life complex systems with stochastic processes cannot be precisely described by analytically estimated mathematical models; that's why the modeling becomes the only available way to study their behavior.
- using the modeling techniques it is easier to provide much more effective control of experiment conditions than just experimenting with the system itself.
- modeling allows to study the long period of system operation in a short period of time;

J. Forrester, a professor at Massachusetts Institute of Technology, almost 50 years ago proposed a new idea of simulation, according to which complex systems should be modeled on the highest level of abstraction, when researcher abstracts from individual objects of the system and examine only the aggregate indicators of such objects, their interaction and interdependence in dynamics [6, 7]. Modeling paradigm where the systems are built researched charts causalities and global influence of some parameters on the other parameters over time, and then created on the basis of those diagrams on the computer model simulates, has been named system dynamics. In fact, this kind of modeling over all other paradigms to understand the essence of what is happening helps identify cause-effect relationships between objects and phenomena.

Despite such simplification, as aggregation, abstraction of individual characteristics, behavior and physical characteristics of the environment in which the process flow, system dynamics model proved quite productive for the study of many difficult problems. Models of business processes, city development, production and the dynamics of population, environment and development model of the epidemic - are just some of the areas of simulation, which copes with the system dynamics.

Nowadays System Dynamics approach to simulation is widely used in the different areas of business [8, 9, 10, 11].

## 4. Methodology

The concept of the new methodology is based on the fusion of the two existing methodologies - Balanced Scorecard (BSC) and System Dynamics (SD).

That kind of fusion in the proposed concept significantly improves both the process of model verification based on historical data and the process of strategic planning (as it is shown in [13] on SD and GQM fusion example).

Workflow - is a sequence of performed actions or a hierarchy of activity.

The Dynamic Enterprise Management principle of operation is implemented as a workflow concept, which includes the following stages of work with DEM projects .

Following the establishment of a new draft the analyst builds up an enterprise model "as is" in terms of system dynamics, then performs validation of the model according to the real-world data provided by the company: the model should be specified until the level of correctness becomes sufficient for the continuation of the work - checking all possible scenarios. Once the model is validated and enterprise development scenarios are set up, the expert chooses the best scenario (business plan) which meets specified business objectives. For example, the optimal scenario is the development scenario, in which the company remains resistant to the effects of the environment throughout its development process.

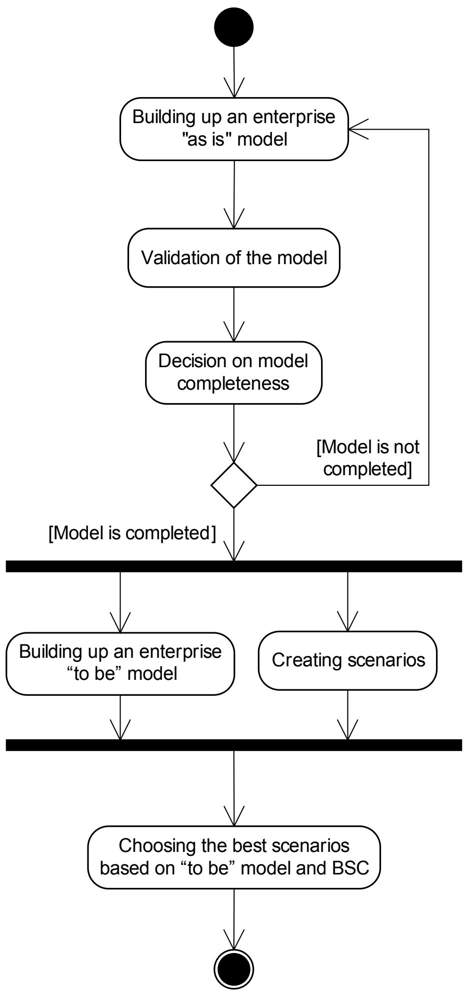

Building up an enterprise "as is" model
Validation of the model
Decision on model completeness
[Model is not completed]
[Model is completed]
Building up an enterprise “to be” model
Creating scenarios
Choosing the best scenarios based on “to be” model and BSC

Figure 2. DEM Workflow

## 5. DEM design

According to the methodology described above, the information system (IS) DEM can be designed (figure 2).

*System Dynamic Tool* – is computer simulation software, where the structure of existing business processes of the company is built on; also different scenarios are simulated with the usage of a system dynamic tool.

*OLAP* (online analytical processing, analytical processing in real time) – is the technology of processing information, that also includes dynamic processes of drawing up and publishing records and documents. OLAP Technology serves, in particular, to prepare business records. Thus, the use of OLAP is necessary for the implementation of the software product Dynamic Enterprise Management.

*Data Management* layer includes *Data Warehouse*, integration buses and a set of protocols for the transfer of data from/to it.

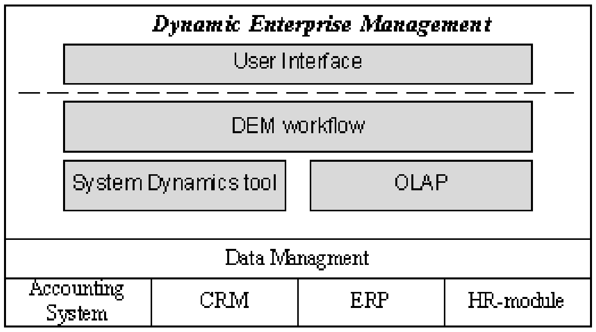


Figure 3. DEM design

In *Data Warehouse* all necessary information is gathered from various sources. Before data get to Data Warehouse, it should be properly processed. Enterprise databases of corporate systems, on the basis of which the Data Warehouse is built, will be called "transactional". Different departments may use different corporate information systems with their own transactional databases. Accordingly, before using these disparate data, they should be examined.

Here comes the question: how necessary information can be accumulated in Data Warehouse, that is how it is possible to transfer the data from different sources and different data formats. In order to be able to perform that kind of integration, a special integration bus is created, through which "communication" of data warehouse with a variety data sources of innovation is fulfilled.

As can be seen from Figure 2, at the bottom layer of DEM architecture there are many sources of data:

*HR-module* - is automated system staff management. *Accounting System* - a class of systems that perform accounting operations.

*CRM* - Customer Relationship Management, System of Management Customer Relationship

*ERP* - Enterprise Resource Planning, systems of planning and company resource management.

## 6. DEM prototype

Currently, a prototype of DEM informational system is being developed system. The language of implementation is C #. As a platform for the prototype implementation the following Microsoft products and technologies were selected: Visual Studio. NET 2008, using the. NET Framework 3.5, MS SQL Server 2000, ASP. NET. Data integration is implemented using MS BizTalk product.

Three components were implemented:

• *DB connector* – is a component that provides a set of classes for connecting to the database DEM;

• *Powersim connector* – is a component that implements a set of classes for working with the simulation tool Powersim Studio;

• *1C connector* – is a component to connect to the database.

The prototype has a web-based user interface (implemented via ASP.NET).

During the prototyping phase it became clear that the development of scenario system planning should start with some certain project. Due to the rich knowledge in software engineering and business connections of the project team it was decided to develop DEM IS in application to strategic planning in software project management.

## 7. Software Process Simulation

There are some critical problems of strategic planning in the area of software development project management and in the area of project portfolio management: going out of time and money scope, closure of software development projects [12]

According to the research reports software companies are interested in the system of this kind. Furthermore to qualify for CMMI level 5 focusing on the process optimization is needed.

Simulation of software processes is the best approach here [13].

The number of simulation models for the software development processes exist [13, 14, 15], the review of the majority of models is made it [13]. The latest tendencies are described in [16].

Attempts to the simulation for the software processes are described in [17]. However there is no composite system for strategic planning in software development.

## 8. Conclusion

In this work Dynamic Enterprise Management, an informational system for strategic planning, and its methodology were considered. DEM IS enables an expert (analyst) to estimate different development strategies, thus, to choose one, the most suitable strategy, from many. In our view, the system meets the modern, ever-changing business requirements. In addition, analysis of the existing strategic planning methodologies was carried out, and it showed that the DEM is a competitive system.

Implementation of the proposed system is realistic, as it is described in the architecture of IS Dynamic Enterprise Management. Currently prototype of DEM system is being developed within Software Engineering Department of Higher School of Economics. Some results have already been obtained in this field – the very first version of the prototype has been already developed.

## 9. References

[1] R. S. Kaplan, D. P. Norton, "*The Balanced Scorecard: Translating Strategy into Action*", Harvard Business School Press, New York, 1996.

[2] Basili V R, "*Software modeling and measurement: The goal/question/metric paradigm",* Technical Report, CS-TR-2956, Department of Computer Science, University of Maryland, College Park, MD 20742, September 1992.

[3] Basili, Victor; Gianluigi Caldiera, H. Dieter Rombach, "*The Goal Question Metric Approach*", 1994.

[4] L Buglione, A Abran, "*Balanced scorecard and GQM: what are the differences?", Proceedings FESMA/AEMES Conference*, 2000.

[5] Averill Law, W David Kelton, *Simulation Modeling and Analysis. Third edition*, McGraw-Hill.

[6] Jay Wright Forrester, *Industrial Dynamics,* Pegasus Communications, 1961.

[7] Jay Wright Forrester, *Principles of Systems.* Pegasus Communications, 1968.

[8] Matthias Ruth, Bruce Hannon, Jay W. Forrester, *Modeling Dynamic Economic Systems (Modeling Dynamic Systems),* Springer; May 13, 1997

[9] John D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*, McGraw Hill Higher Education, December 1, 2000

[10] Bernard McGarvey, Bruce Hannon, Dynamic *Modeling for Business Management: An Introduction (Modeling Dynamic Systems); 1 edition,* Springer, January 8, 2004

[11] Kim Warren, *Strategic Management Dynamics*, Wiley, February 8, 2008

[12] Standish Group International Report, 2004

[13] Raymond J. Madachy, *Software process dynamics,* Wiley-interscience a john wiley & sons, inc., publication ieee press, 2008

[14] Abdel-Hamid T. and Madnick S., *Software Project Dynamics*, Englewood Cliffs, NJ: Prentice-Hall, 1991.

[15] Acuna S., Juristo N., Moreno A., and Mon A., A *Software Process Model Handbook for Incorporating People's Capabilities*, 2005.

[16] Acuna S., Sanchez-Segura M., *Series on Software Engineering and Knowledge Engineering Vol. 18. New Trends in Software Process Modeling,* World Scientific Publishing, 2006.

[17] Seunghun Park, Hyeonjeong Kim, Dongwon Kang, Doo-Hwan Bae. *Developing a Simulation Model Using a SPEM-Based Process Model and Analytical Models,* 2008.

# Simulating genes operation and interaction

Rekubratskiy V.A.
*Centre "Bioengineering" RAS*
*vrecobra@gmail.com*

Korotkova M.A.
*Moscow Physical Engineering Institute*

## Abstract

*Gene operation and interaction occur in every organism. Gene network is commonly used to describe gene interaction issues due to its complicity. Simulation of gene networks operation is an essential problem of bioinformatics. System providing tools for creating, setting up and simulating gene networks is described. Broad simulation abilities along with high extensibility are system's advantages to be noted.*

## 1. Introduction

Gene operating and interaction determines development of every organism on the Earth [1]. Though many genomes are well examined nowadays [2], gene interactions issue still remains a subject of intensive investigation. The main point is that interactions can often be revealed only through indirect indications, such as gene activity changes [3], [4]. Activity of some genes can have positive or negative influence on other genes' activity [3]. This influence can be detected during experiments. Many articles with assumption about certain gene relationships are published every year [5]-[9]. Gene network is represented by directed graph with vertices corresponding to genes and arcs corresponding to positive and negative interactions. It is commonly used to describe gene interactions of an organism [10]. Gene network is usually worked out by a biologist studying interactions issue. Unfortunately, a network based on some experimental data may contradict the data received later or by another researcher (i.e. works [4] and [7] to be compared). Several distinct gene networks are usually possible to be drawn up basing on incomprehensive data set also [11]. To find the most accurate gene network is required a network verification using some other experimental data.

Computer system providing simulation of gene interaction and operation is needed for comparing and testing gene networks. Such a system may give the possibility for automatic verification of gene network conformity with experimental data given.

## 2. Simulation system

### 2.1. Gene model

In real organisms, a gene can be active in some cells and inactive in others. The degree of activity may also vary. Each gene is responsible for producing its special protein [1]. A gene is considered active in a cell when some amount of corresponding protein is found in that cell. Amount of gene's protein can be measured in real cells and thus can be used for verifying gene network conformity. Thereby protein amount is an essential characteristic for gene model. Protein production rate depends on different factors and may vary from cell to cell and from gene to gene. The amount of other genes' proteins is one of the factors influencing the protein producing rate and this gene parameter used for gene interaction modelling. Gene networks are often simulated using Petri networks [12], [13]. Only one gene characteristic – amount of corresponding protein – is taken into account within this approach. However, a model taking some other gene parameters into consideration could express more compound dependencies of gene interactions.

The simulation system proposed takes both gene interactions and states into account. Each gene in the model is described by three parameters: product, activity and block. The first one – product – describes the amount of gene's corresponding protein. It also represents gene ability operate on other genes and thus can be defined as gene outer state. The second parameter – activity – represents gene ability to generate its product and thus can be defined as gene inner state. The speed of product generation may vary. Activity function is introduced as another gene characteristic to reflect generation speed variability. Both product and activity are real positive numbers, in contrast to the third Boolean parameter. This parameter was introduced in the model to express the fact that some genes actually may start or stop operating at any stage of organism development process. When gene is blocked (i.e. its block parameter is true) it cannot nor generate any product.

In nature, product of every gene undergoes the degradation process, i.e. its value gradually decreases if gene activity falls to zero. This fact is also taken into account in the simulation system proposed. The speed of degradation expressed by degradation function is another gene characteristic.

### 2.2. Gene interactions model

Each interaction in the simulation system is described by a set of rules. These rules identify the conditions of the interaction arising and how gene parameters are modified through the act of interaction. Rules defining necessary conditions of interaction are expressed by inequalities and the ones defining parameters change expressed by the equations. Algebraic expressions including any combinations of real numbers and gene parameters can be used in both groups of rules. Commonly, gene product parameters alone are used in conditions of interaction reflecting the fact that product is a characteristic of gene interaction ability.

### 2.3. Simulation process

Gene network operation is simulated as step-by-step process. Check of active interactions is performed on each step. Changes to gene parameters are not applied until an interaction was executed. Instead, all changes are accumulated to be applied at the end of the step. Changes of gene product values depend on gene product generation ability (i.e. activity), so those changes are made at the end of each step also but before applying parameter changes.

## 3. Implementation methods

Simulation system was implemented using Microsoft Visual C++ as a Windows application. The application enables a user to create a gene network by inserting genes into system, drawing interactions scheme and setting up some numeric parameters. After network was created and its initial state was set up, a user can monitor overall simulation process (on network scheme and parameters table) and particular gene activity changes (on graph of function).

### 3.1. Gene network implementation

Gene network is implemented as a two-level system. The first (outer) level is used for intercommunication with a user. It enables him to set up network topology and gene parameters and then monitor gene states during simulation process. The second (inner) level is used to simulate network operation. First, all the information set on the outer level is transferred to the inner one. Simulation step is executed via the second level and then results are translated back to the first level, and so on.

### 3.2. Network editor

The system provides a graphic user interface for setting up gene network. Network topology drawing has a lot in common with working in many scalar or vector graphics editors. Arcs corresponding to gene interactions are created similar to lines drawing, genes are repositioned by mouse pointer, etc.

Extensive class system was used to implement network edition routines. All network objects are derived from common base class containing information about object’s inputs and outputs, its position, id, etc. It defines an interface for operations upon network objects as well. These operations include saving, loading, setting up, visualizing and some others. Many actions and checks could be executed uniformly for vertices (genes) and arcs (interactions) due to this design. It gives rise to two related class hierarchies: one for network objects (genes, interactions) and one for operations upon objects. In terms of design patterns, a combination of “Composite” and “Visitor” patterns is used providing very flexible class architecture.

### 3.3. Extensibility

System simulating single gene network operation was initially developed. After that is was extended to perform simulation of multicellular organism development. The extension was made easily due to the system design providing for it. New network object types can be easily introduced on the outer network level, e.g. objects corresponding to some cell activity processes or to environment influence. One new object type was introduced during system expanding to provide creation of processes for cell parameters modifying. The inner network level provides uniform parameter reference system. It enabled simple introducing of new parameters, such as cell size and position, equally with existing gene parameters. The further system extension towards more comprehensive model is still possible.

## 4. Novelty and results

Quite a few software systems for gene network simulation have been developed so far [14], [15]. Unfortunately, these systems simulate operation of a standalone network without its connection to cells life activity. In fact, there may by observed genes interactions of genes of the remote cells with different gene states in each one. These situations could hardly be described by a single gene network models.

There are only few software systems simulating multicellular organism operation that can take gene interactions into account [16]. The using of this model requires some special knowledge in programming.

The system proposed is designed to be a standalone application that does not require any other mathematical or simulation software running. It grants user a simple to use tool for drawing his own gene networks. It also provides a set of relatively simple algebraic rules for describing gene interaction details. This small set of rules, however, is shown to enable simulation of rather wide range of gene and cell interaction processes. Some processes of genetic control for Arabidopsis thaliana development, as well as for HCV infecting and developing, were successfully simulated using the designed system.

## 5. Conclusions

The simulation system and its software implementation were designed. The simulation system provides a comprehensive gene operation model and simulation of intercellular gene interactions. The software implementation enables a user to easily draw a gene network and describe intercellular gene interaction details by a set of relatively simple algebraic rules. The whole system is shown to be an extensible one and enable simulation of rather wide range of gene and cell interaction processes.

The further development is intended to make system interface simpler for a biologist providing an easier description of gene interaction processes. A more comprehensive model of organism operation is to be designed further taking advantage of system extensibility.

## 6. Support

The work is supported by Science & Technology International Park "Technopark in Moskvorechje" along the “UMNIK” program.

## 7. References

[1] A.S. Konichev, G.A. Sevastyanova. “Molecular biology”, “Academia” publishing center, Moscow, 2003

[2] National Center for Biotechnology Information: http://www.ncbi.nlm.nih.gov/

[3] N.A. Kolchanov, E.A. Ananko, F.A. Kolpakov et al. “Gene networks”, *Mol. Biol.*, Moscow, Russia, 2000, July-August, 34(4), pp. 449-460

[4] J.L. Bowman, G.N. Drews, E.M. Meyerowitz. “Expression of the Arabidopsis floral homeotic gene AGAMOUS is restricted to specific cell types late in flower development”, *Plant Cell*, American Society of Plant Physiologists, United States, 1991, August (8), pp. 749-758

[5] T. Gedeon, E.D. Sontaq. “Oscillations in multi-stable monotone systems with slowly varying feedback”, *J Differ Equ*, Elsevier, Netherlands, 2007, August 15, 239(2), pp. 273-295

[6] G. Pogorelko, O. Fursova, E. Klimov. “IDentification and Analysis of the Arabidopsis Thaliana Atfas4 Gene Whose Overexpression Results in the Development of A Fasciated Stem”, *J Proteomics Bioinform*, OMICS Publishing Group, United States, 2008, October, 1(7), pp. 329-335

[7] S.V. Shestakov, A.A. Penin, M.D. Logacheva, T.A. Ezhova. “New modified scheme of genetic control of flower development”, *Alive systems technology*, Moscow, Russia, 2005, V. 2, №1, pp. 37-46.

[8] C.Capelli, F. Brisighelli, F. Scarnicci, A. Blanco-Verea, M. Brion, V.L. Pascali. “Phylogenetic evidence for multiple independent duplication events at the DYS19 locus”, *Forensic Sci Int Genet*, Elsevier, Netherlands, 2007, December, 1(3-4), pp. 287-290

[9] B. Zheng, X. Lu. “Using protein-semantic network metrics to evaluate functional coherence of protein groups”, *AMIA Annu Symp Proc*, American Medical Informatics Association, United States, 2007, October, 11, pp. 1174

[10] N.Geard, J. Wiles. “A gene network model for developing cell lineages”, *Artif. Life*, MIT Press, United States, 2005, Summer, 11(3), pp. 249-267

[11] C. Gustafson-Brown, B. Savidge, MF Yanofsky. “Regulation of the arabidopsis floral homeotic gene APETALA1”, *Cell*, Cell Press, United States, 1994, January 14, 76 (1), pp. 131-143

[12] S. Grunwald, A. Speer, J. Ackermann, I. Koch. “Petri net modelling of gene regulation of the Duchenne muscular dystrophy”, *Biosystems*, Elsevier Science Ireland, Ireland, 2008, May, 92(2), pp. 189-205

[13] L.J. Steggles, R. Banks, O. Shaw, A. Wipat. “Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach”, *Bioinformatics*, Oxford University Press, England, 2007, February, 1, 23(3), pp. 336-343

[14] "Ingeneue: gene network simulation software" http://rusty.fhl.washington.edu/ingeneue/index.html
[15] "GeNESiS: gene network evolution simulation software" http://genomics.iab.keio.ac.jp/genesis.html
[16] "CellModeller Project" http://www.archiroot.org.uk/doku.php/navigation/cellmodeller

# Information system user interfaces automatic creation

Alexander Korotkov
Moscow Engineering Physics Institute
Moscow, Russia
email: aekorotkov@gmail.com

***Abstract*—Very actual task for the information system is automatically user interface creating. Solution of this task greatly decreases the information system development time. There are various approaches for automatic creation of user interfaces. In this article most popular approaches are surveyed and their problems are described. Author suggests the hybrid approach which allows minimizing the problems of existing approaches.**

***Index Terms*—user interfaces, code generation, information systems, databases, objects inheritance**

## I. Introduction

The programming of some kinds of applications from scratch is not reasonable. The continuous extension of the libraries is widely used. The libraries are designed to solve tasks that are isolated from general applications logic. That's why the approaches like solution of general tasks with complex configuration and code generation is also popular.

The user interfaces (UI) creation is the one of the tasks when programming from scratch is unreasonable. The rich UI libraries solve only part of task. Some meta-information that may be helpful to create UI already exists as the rule. It may be database structure for example. On the one hand, UI very frequently needs some project specific features. On other hand, many UI parts are evidently reasonable for automatic creation. The problem is to develop approach which allows to automatically creating UI which can be extended by project specific features.

The various approaches of the automatic creation UI for information systems currently exist. Two approaches are the most popular. The first approach uses the program code generation which can be modified by programmer in a future. The second approach uses runtime generation UI which is based on the meta-information and the user settings probably. Let's consider these approaches in details.

## II. Existing approaches

The code generation based on the meta-information is popular practice. This approach is automation of the process which programmers are making manually in other case. When programmer has the task to develop UI for some object he should already has encountered with all parts of this task in some degree. He will copy the parts of a code from his old projects, correct identifiers and do some other correcting to make it working together. Then first version will be done. When we use the code generation most part of these actions are executed automatically (Fig. 1). Automatic code generation eliminates the nasty errors from lack of attention. The problem of this approach is a lot of iterations that development process has usually. These iterations may contain meta-information correction. Used for the code generating the meta-information may changes also. Then there is question how to transfer these changes to already generated and corrected by a programmer code. We can generate the UI code again and manually correct it again (Fig. 2). Otherwise we can manually correct UI code due to meta-information changes (Fig. 3). In any case manually correction is needed and this correction has a not minimal volume.

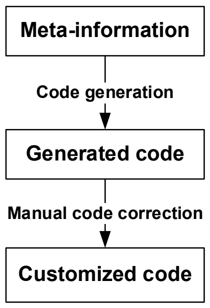


Fig. 1. The scheme of the automatic UI code generation

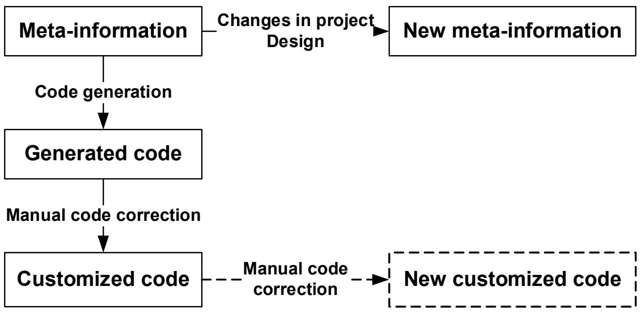


Fig. 2. The scheme of the automatic UI code generation for meta-information changing (the first alternative)

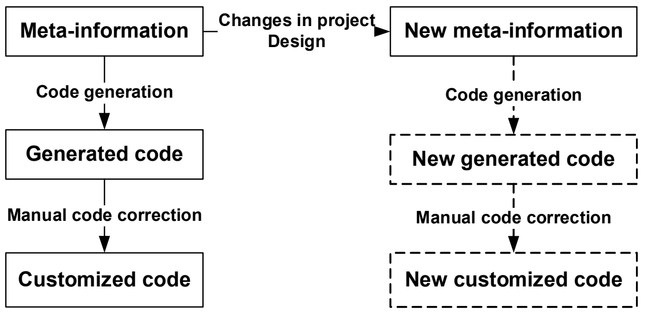


Fig. 3. The scheme of the automatic UI code generation for meta-information changing (the second alternative)

The second approach doesn't use the manual code correction and includes all information needed to configure UI into meta-information. In this approach we have not problem of transferring changes from previous approach because manual generated code correction is not used there.

Databases administration tools use a simple implementation of this approach. These tools provide a viewing and editing interface for each table of a database. The more complex example is the administration generator of Symfony PHP Framework [1].

In this approach any specific UI customizations must be covered by the meta-information and generator possibilities. So if some new sort of UI customization is needed then meta-information and generator must be extended to cover such customization.

The most serious problem of this approach is that reprogramming a UI generator is frequently needed. It may produce the new generator functions to be appropriate to specific project needs but doesn't solve the general problems. In this case the generator design may suffer.

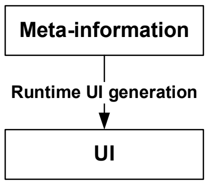


Fig. 4. The scheme of the runtime UI generation.

## III. Proposed approach

Author proposes the hybrid approach when generated code and manual corrections are logically separated. When the meta-information is changed than the generated code will be generated again but existing manual corrections do not require any changes or these changes will be minimal. To implement this approach the two tasks should be solved:

1) Code generator should be flexible enough to generate the application which has a skeleton which doesn't require the changes during the customization. Because it would be problematical to implement logically separation of manual corrections which include the application restructure.
2) The mechanism of separation of generated application and manual changes should be found.

For the separation of the generated application and the manual changes the inheritance mechanism is proposed. In this approach for each automatically created UI component the two classes are generated (Fig. 5). The first class is automatically generated UI component. The second class inherits the first class and it is empty initially. Programmer can manually fulfill the second class for UI component customization. When meta-information is changed than the first class will be regenerated but second class do not require any changes or these changes will be minimal and dealing with the manually added features (Fig. 6). This approach is very similar to popular approach in the Object Relation Mapping (ORM) [2] software where base classes of persistent objects are generated and derived class can be manually customized. Let's consider how this approach can be applied to very usual UI components of information system such as the grid and form.

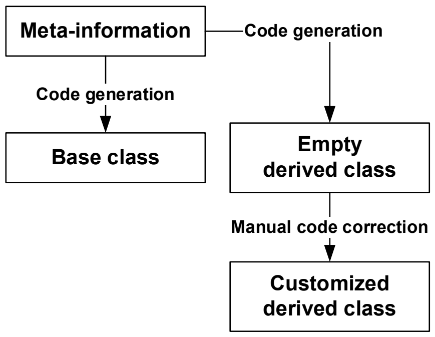


Fig. 5. The scheme of the hybrid approach of a UI generation

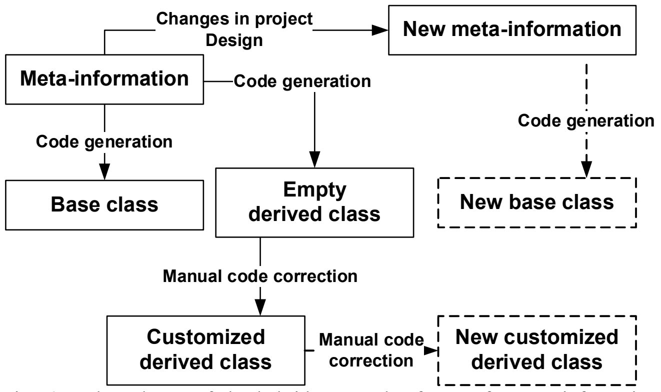


Fig. 6. The scheme of the hybrid approach of a UI for meta-information changing

## IV. Implementation

To implement this approach the Web-interface which uses the Javascrit-framework ExtJS [3] is generated. The generated class for gird is derived class from the Ext.Grid. This class contains configuration of columns, render methods of columns, appearance configurations methods of columns and UI objects for cell editing. The derived class can redefine these configurations, methods and objects. Objects in base class are defined by the configurations without the explicit constructor call. It helps to avoid the excess of the objects creation when they are redefined in the derived class. The generated class for form inherits Ext.FormPanel. This class contains UI objects using for editing of the form data. The derived class can redefine these objects. Similar to the grid base class the objects are defined by their configurations. Derived classes can contain additional methods and properties which implements extended inner logic.

## V. Conclusion

The one of problems of automatic UI creation is the developing approach which allows automatically creating UI which may be extended by the project specific features. Various approaches for automatic creation of user interfaces (UI) for information systems currently exist. The one of these approaches uses the program code generation. There is a problem in this approach when meta-information is changed. Then manually changes are required. These changes may concern the one more UI component customization or the manually transfer of the meta-information changes to program code. Other approach is runtime UI generation. The most serious problem of this approach is that interface is strictly limited by laying therein customization possibilities. Reprogramming some feature of such UI is difficult. Author proposes the mixed approach when inheritance is used to separate generated application and manual changes. The two classes are generated for each UI component. The first class is automatically generated UI component. The second class inherits the first class and it is initially empty. Programmer can manually customize the second class. When meta-information is changed the base class is regenerated only and the manual changes in derived class are minimal. This mixed approach make possibility for the minimizing of the problems of the surveyed approaches for the automatic UI generation. This approach was proved to be functional.

## References

[1] Symfony Open-Source PHP Web Framework, http://www.symfony-project.org/

[2] Propel ORM PHP framework, http://propel.phpdb.org/trac/

[3] Symfony Admin Generator, http://www.symfony-project.org/book/1_2/14-Generators#Administration