Background Information

Generative Programming - From Theory to Practice

THIS DOCUMENT COMPRISES THE FOLLOWING INFORMATION:

- *What does Generative Programming mean*
- *The Generator Principle*
- *ANGIE – A New Generator Engine*
- *Generative Programming in Practice*

ABSTRACT:

The term "Generative Programming" first arose about five years ago. It coalescences a lot of different current concepts, that are mainly focused on a higher degree of automation in software development, as well as the consistent development for reuse. While these considerations were predominantly of theoretical nature in the beginning, nowadays practicable solutions is what everyone is looking for. Delta Software Technology Group, into the development of generative tools for many years, realised some of these concepts consequently and is about to develop them further. Result of this maturity is a kind of a modular system for generators with an innovative "Generator Engine" as its core. Meanwhile, these new tools are proving themselves in challenging projects and the perspectives for the further enhancement of these solutions are obvious to see.

1 WHAT DOES GENERATIVE PROGRAMMING MEAN?

The usage of software serves the automation of processes. This is why the fact, that the complex processes of the software creation itself can hardly be automated, takes one by surprise.

Fundamentally, the deployment of software in order to design software is apparent. On the other hand, fully automatic generation of entire application systems cannot be achieved in a short-range; moreover, it seems to be implausible to be attained in the future as well. Below this high demand, many potential variants of tools are imaginable that are able to create software systems and components in a more or less automatically manner. The automation of software production is not only to serve the speeding up of software development and the reduction of the development costs; it furthermore improves the quality as well as it causes less error liability. Finally, yet importantly, maintenance expenditure and the costs resulting from it will be also reduced.

Of course, these considerations aren't new, but somehow in between, the idea arose, that the usage of object-oriented techniques became superfluously. Anyway, on the one hand, the object-oriented programming triggers off, if used correctly, a higher degree of abstraction but, on the other hand, hardly provides automation. This can be verified e.g., in the widely spread design pattern by Gamma [GHJV 95]. They became a de facto standard for most of the OO- programmer and it can be assumed, that they are used thousandfold. But what its really about is the question, how can this innovative and challenging principle be implemented individually? Hardly ever by the usage of tools and, for sure, the object-orientation itself does not provide any help.

The preferred tool is still CPA (copy-paste-adapt) – this is downright barefoot software development.

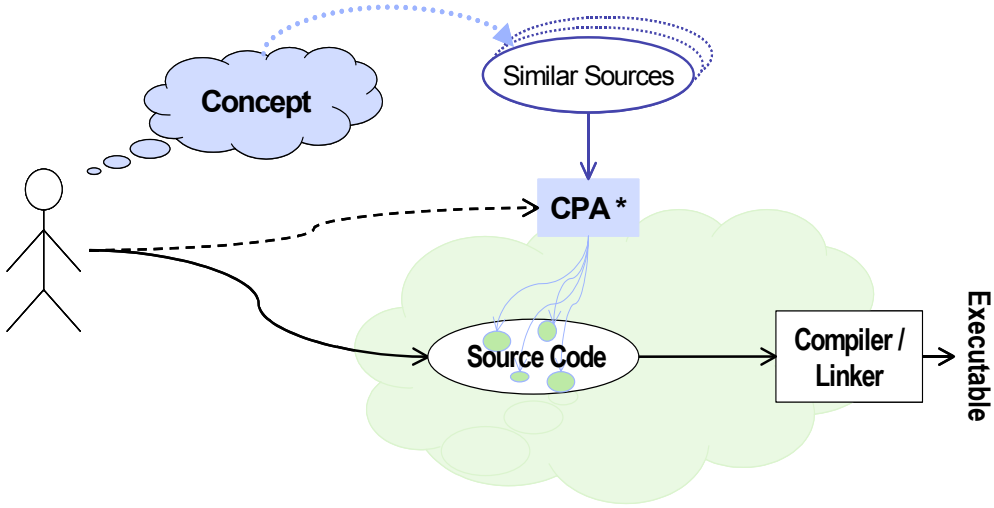


Fig 1: The „Barefoot Method“- Copy-Paste-Adapt

Hence, it does not astonish anyone at all that throughout the last few years the concepts of automated programming became increasingly significant.

The basic idea is represented by the following quotation [CE99] in an utmost simple manner:

> *The automation assumption:*
>
> *"If you can compose components manually, you can also automate this process"*

In this context, the term *Generative Programming* (GP) arose for the first time about five years ago. Last year, the same-named book by *Czarnecki* and *Eisenecker* was published [CE00].

GP is considered to be a new paradigm in software development that does not compulsory compete with the current paradigms, especially with the object-orientation, but supplements them. Besides the automation of software development, it is mainly the realisation of software families, which is in the focus of the contemplation. Conventional development is about developing components or even entire systems with similar characteristics individually. For the most parts in these cases, there is only the previously referenced, ancient CPA method available – even on design level. On the contrary, the GP principle assumes that it is possible to generate the members of a system family, namely on the base of a common model of the system family, the *Generative Domain Model*.

This model consists of three elements:

1.A method to specify the members of a family; 2.modules that enable the creation of each member and 3.the configuration know-how to transcribe specifications into implementations.

This is similar to ordering a car: There is an order form to purchase a car, components from which the car is assembled of and somebody who knows the way the car can be built as per order.

Instead of the term *system family*, the term *product line*, which is of slightly different meaning, is being used as well. This difference is not relevant for the considerations presented in here, so the term

Modelling and specification of system families is processed with techniques such as *Domain Engineering*, *Feature Modelling* and *Domain Specific Languages* (DSL's). We will not go deeper into that here.

The realisation of the modules and the implementation of the configurations, i.e. the transcription of the specifications into software systems and components require adequate generative tools. The concept of a new generator tool, resp. of an entire tool family and the experiences gained with it will be presented subsequently.

2 The Generator Principle

The main task of a generator is to translate specifications of a higher abstraction level into a lower level format, which for the most part means into a source code for a compiler. To that end, the generator has to be familiar with the concept to be implemented; so to say, it has to contain configuration knowledge.

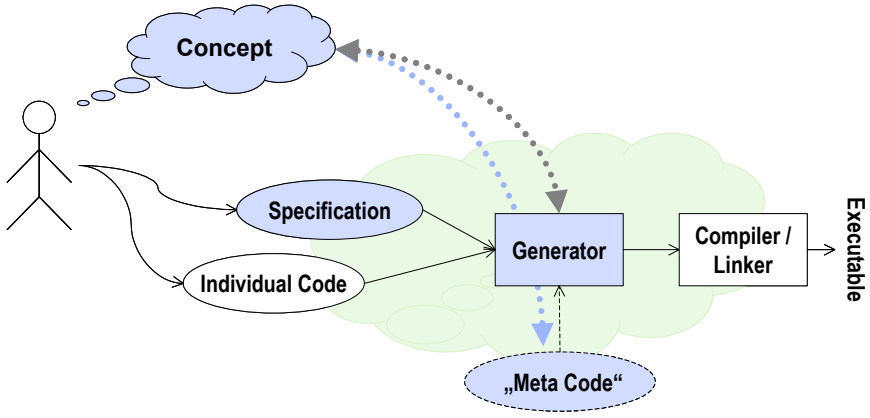


Fig 2: the Generator Principle

There are many different ways to realise this within a concrete generator. Many generators are specialised for particular tasks. For example: GUI Wizards, DBMS pre-compilers or UML transformers. These generators work highly efficient in their areas of responsibilities.

On the other hand, due to their specialisation, they are not very flexible and cannot be adapted to individual tasks, which on the other hand is one main condition for the GP.

Another generator approach are macro processors or template mechanisms in compilers. They are a sort of "Do-it-yourself" generators with high flexibility, that attributes to their principle; but without any individual intelligence, which causes a great deal of usage expenditure.

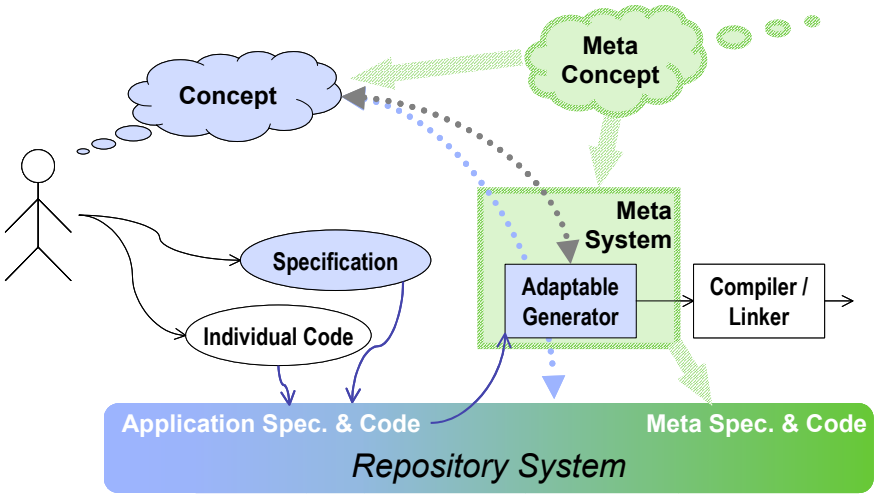


Fig 3: the updated Generator Principle

In addition to this, there are generators that are a combination of those two principles. They comprise pre-defined specialised core functions for various task designations on the one hand, and are individually extensible by macro modules on the other hand. *Delta Software Technology* used this principle for its generators ever since.

With the intention of being successfully used in a GP environment, real innovative generators have to comprise even more efficient performance.

Firstly, they have to be equipped with the ability to be continuously adapted and configured. To that end, a configurable generator infrastructure is required in order to provide individual generators for any system families without the need to fall back on the tool vendor.

A real generator meta-system with tools for the development and adaptation of generators, based on a repository system is required for it. The meta-specifications as well as the application specifications are being managed in this repository.

3 ANGIE – A NEW GENERATOR ENGINE

After several years of analysis and general conception, in 1999 the development of a new generator technique began. With it, a component-based modular system for generative programming was to be designed instead of the average monolithic generators with specialised tasks.

The first and most important component that was developed is a new "Generator Engine" called *ANGIE*. The partially new characteristics to be pointed out are: the frame-based architecture, the extensible generator language, the generator technique as a component and, as well, an XML repository basis.

3. 1 A Frame-Based Generator Architecture

The Frame/Slot approach origins from the AI and has been introduced during the 70ties in the scope of pattern recognition. Later it proved that this procedure could also be used for the analysis and synthesis of languages.

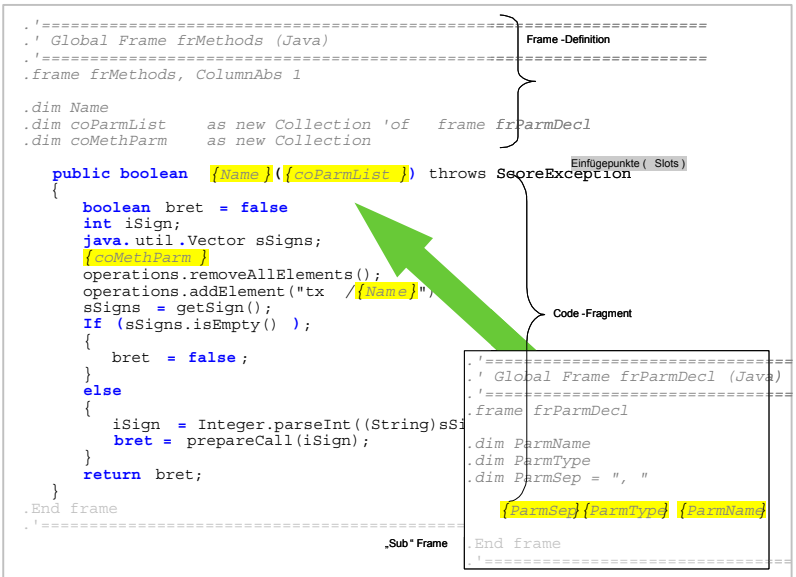


Fig 4: ANGIE Frames and Slots

Delta Software Technology already used this standard on programming languages ten years ago. For instance, the Y2K problem was tackled with tools based on this technique. Extensive program systems have been examined and corrected – amongst them were programs with hundreds of thousands (!) code lines. Some other time interfaces of legacy applications have been interpreted and converted into object-oriented implementations automatically.

Frames are code fragments, similar to classes and objects in object-oriented languages. Any number of copies (instances) can be created and being modified as well as being stored in a repository in the course of the generation process. Slots within the frames receive names or variables and additional sub-frames. This leads to tree structures, alike abstract syntax trees, with any depth and complexity. They can be created, evaluated and modified by generator scripts. These scripts do only have to know the frames and frame variables; the content of the frames (the code fragment) is usually invisible for the scripts. During the final export, the abstract memory and repository contents will become concrete source code. In general, it is sufficient to exchange the frame definitions in order to support another programming language by the usage of the same set of generators and scripts.

3.2 The Extensible Generator Language

The separation of the constructing process of (code) production, achieved trough the frame concept, opens up new horizons in the realisation of generators. However, prerequisite is an efficient generator language that has to provide more options than only the common template or macro mechanic.

The script language of *ANGIE* is a full-blown programming language, especially designed for the development, enhancement and configuration of generators; so to say, a generator DSL. The language is syntactically related to Visual Basic, but extended by frames as object-oriented constructs. The most important elements of the language are:

- Modularisation with declarations- and code modules, script functions, variable containers and frame declarations
- Global and local data declarations, statically and dynamically, arrays and collections (with keys)
- All necessary imperative instructions, assignments and function calls
- Intrinsic functions, e.g. for the string handling

The script language is realised as an "embedded language". This means for instance, script instructions recognizable by a particular flag, can be interspersed into the coding of the target language without the constraint of altering it. Besides, a special editor available identifies the *ANGIE* constructs and displays the difference between script and target language chromatically (see also figure 4).

As a distinction to ordinary generator systems there are not two different languages for the development of the core functions (generators or processors) and for the configuration and adaptation (macros, templates and the like) required. The script language is both: uncomplicated and powerful so it can be used for all levels equally. As a result of compiling the *ANGIE* scripts, a loss of performance is being avoided right from the very beginning.

In cooperation with the University of Paderborn, an efficient compiler has been developed. It runs so fast that it compiles most of the scripts just-in-time.

In addition to prefabricated generators, being created from compiled scripts and also pre- defined standard scripts, which can be adapted to the individual projects, you are of course provided with the possibility to supplement any individual scripts to the projects as well. The entire system is focussed on extensibility. With it, there are three levels of extensions:

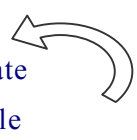
- Compiled ANGIE script functions can be treated as AddIns that hardly differ from the intrinsic functions afterwards.

- Via a particular interface, External functions may also be implemented as AddIns being programmed e.g. in C. There are already AddIns for the linkage to data and repository interfaces available

- Even the realisation of intrinsic functions has been simplified by a standardised and "tight" interface so the future further development is not limited at all.

3.3 The Generator Technique as a Component

Most of the "habitual" generators and pre-processors follow a variant of the waterfall model: with it, the generator is a monolithically element in the development process, isolated from all other tools of the development environment:



In comparison, *ANGIE* was designed as a real component that can be integrated into any other tools such as interactive design tools. Hence, the generation function is considered as a service, which can be called on demand whenever there is something to be compiled. This enables a successive and sectional software development, perhaps hidden below an interactive user interface.

Fig. 5: Generator as a Component

In combination with the possibility to differentiate between construction and final code production that arises from the frame architecture, a new more up-to-date process scheme is being created:

3.4 The XML Repository Basis

Storage and exchange of development data in pure textual format are not sufficient any longer. Too many information get lost or have to be filtered out from the text arduously. Consequently, the linkage to a repository is a central element of the new generator system.

All current tools of *Delta Software Technology* use an XML-based repository format. The XML standard enables a smooth exchange of data. Furthermore, there are plenty of tools for evaluation and modification obtainable. The XML interface of *ANGIE* admits the inquiring of information from the repository from the scripts directly and to modify and create them.

The repository linkage is not restricted to the XML format. Any additional interface can be implemented as an *AddIn* (see also 3.2 "The Extensible Generator Language"). Presently an interface is being realised that will support the MOF standard (*Meta Object Facility* of the OMG).

4 PRACTICE

At the first glance, the subject *system family* seems to be something interesting for only a few user groups such as banking institutes, insurance companies or telecommunication enterprises. There are many application systems with a significant number of similar sub-systems, which can be regarded as *system families*. Besides these large-scaled *system families* there are also small-format *system families*. These are for example being created from sub-systems with an identical architectural pattern or programs and modules, which follow a common design pattern. In these cases, the attainable degree of reuse and automation is very high, too.

Yet, with the previous tools of *Delta Software Technology* in particular large-scale users and software houses created extensive systems for a configurable and individually extensible application generation. This led to the realisation of real product lines and system families even tough these terms were not familiar at the time of the creation of most of these applications.

A noteworthy challenge for the new generator technique was the conception of the *Generated Adaptive Frameworks* architecture and their realisation within the integration tool *SCORE/Integration Suite*.

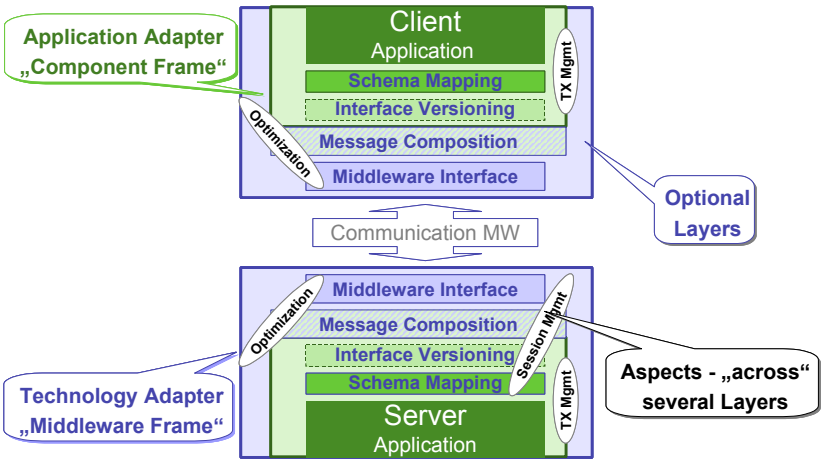


Fig. 6: SCORE/Integration Suite Framework

Central element of the adaptive framework is a layer architecture with strict separation between the necessary semantically and syntactically mappings on the one hand, and the linkage to middleware, system and language platforms on the other hand.

The modules of this framework are generated; on the one hand as separate layers, on the other hand as an implementation of technical aspects to be woven into those layers.

The generators were realised with *ANGIE* and have been afterwards integrated into an interactive tool, the *Component Manager*. An XML repository serves as the information memory (see also chapter 3.4). The principle of a generator as a service, as expounded in section 3.3 "The Generator Technique as a Component" is also used in here. The generator functions are centrally controlled by the *Component Manager* and assist, amongst others, for example:

- Transferring interface descriptions of any origin into a neutral format.
- Supplementing and extending user entries during work automatically, controlled by adaptable scripts
- Generating the entire framework. Whereby, on the base of the same description in the repository, output for the most different purposes and programming languages will derive. Thereby, the Frame/Slot principle can bring all its remarkable advantages to bear.

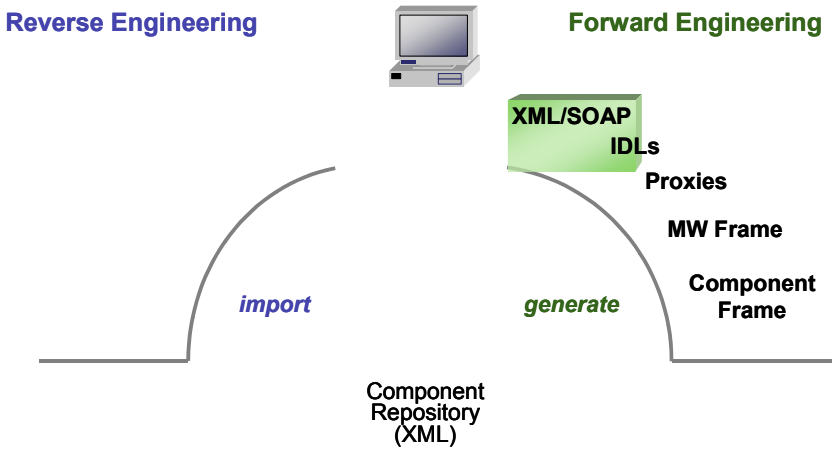


Fig. 7: Tool Structure of SCORE/Integration Suite

The generators may be supplemented by scripts in an almost unlimited way and, as well, be adapted to the individual requirements.

SCORE/Integration Suite, being used in various extensive projects since last year, proves its reliability especially while being used in combination with current object-oriented and component-based technologies like *Java*, *J2EE* and *EJB*.

5 Result and Prospects

Just a few years ago, terms like *Generative Programming*, *System Families* and *Product Lines* were only familiar to a small number of software specialists. Meanwhile, these subjects are a permanent part of high-ranking events such as the OOP in Munich or of the CDUF in Frankfurt. Nowadays, numerous enterprises deal with GP.

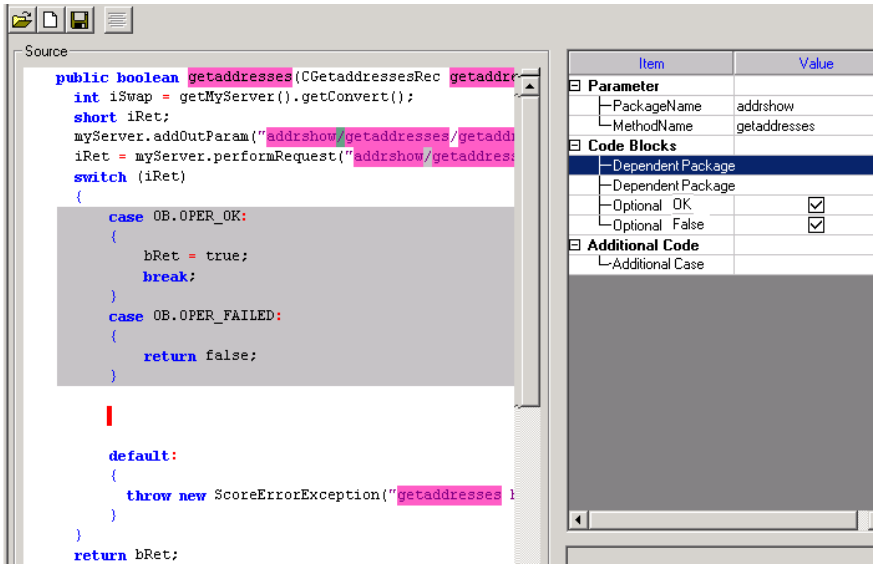


Fig. 8: Sequence of the Pattern By Example Editor

Concurrently to the theoretical and methodical discussions, tools are being designed that help to realise the latest innovative ideas practically. One example is the new generator technique *ANGIE* provided by *Delta Software Technology*. This doesn't mean we've already reached the end of the road but a first important step was done. Primarily it is to be pointed out, that this technique has proven itself within various important projects.

Several further developments are based on these results. Initially it is about supporting the modelling process for system families, e.g. by editors for *Feature Models*. To that end, there is an active cooperation with the advanced technical college of Kaiserslautern.

A second project is focussed on the interactive support of the development and usage of *Code Patterns* via *ANGIE*. The respective tool is based on the *Pattern By Example* concept (PBE).

With it, the idea is to automate the manually CPA process (as explained in chapter 1).

- A code fragment that is to be re- or multiple used, is loaded into a specific editor. Variable and optional areas are marked and equipped with names, conditions and default values. The result is being stored in the repository format.

- Step by step, a library of Code Patterns will emerge. Whenever one of these patterns is to be reused, it can be selected in the PBE tool. The tool verifies the variables or takes care of the pre-settings. The code fragment, expanded from the generator service, can now be implemented into the target program at will.

Gradually, the world of tools for Generative Programming expands. Always with reflections on the practice, so as the previous development results from the combination of current theoretical concepts and many years of experience in designing generators and further software tools.

A light version of this tool is already freely available for download. Please see pbe.d-s-t-g.com.

For more information regarding the subject Generative Programming, the tools, as well as to the freely available beta version of PBE, please visit our Homepage at www.d-s-t-g.com.

LITERATURE

[GHJV 95] E. Gamma, R. Helm, R. Johnson and J. Vlissides: *Design Patterns*. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995

[CE99] K. Czarnecki and U.W. Eisenecker: *Components and Generative Programming*. 7th European Software Engineering Conference, Toulouse, September '99

[CE00] K. Czarnecki and U.W. Eisenecker: *Generative Programming. Methods, Tools and Applications*. Addison-Wesley 2000

CONTACT

Delta Software (Deutschland) GmbH
Novesiastraße 38
D- 41564 Kaarst
Tel: + 49 / 2131 / 51 08 - 0
Fax: +49 / 2131 / 51 08 80
e-mail: info@delta-software.de
www.delta-software.de

Delta Software (Österreich) GmbH
Rennweg 79-81
A- 1030 Wien
Tel: + 43 / 1 / 72 94 36 – 0
Fax: +43 / 1 / 72 82 22 84
e-mail: office@delta-software.at
www.delta-software.at

SAXOS Informatik AG
Industriestraße 6
CH- 8305Dietlikon
Tel: + 41 / 1 / 8 35 20 30
Fax:+ 41 / 1 / 8 35 20 46
e-mail: saxos@saxos.ch
www.saxos.ch