DOLPHIN - A system for compilers development, teach and use

Doutorando: Paulo Jorge Teixeira Matos
Escola Superior de Tecnologia e de Gestão de Bragança
Campus Santa Apolónia, 5300 Bragança
Portugal
pmatos@ipb.pt

Orientador: Pedro Rangel Henriques
Universidade do Minho
4700 Braga
Portugal
prh@di.uminho.pt

Abstract

DOLPHIN is a framework conceived to help the development of compilers. It is based on model of code representation, designated by DOLPHIN Internal code Representation (DIR), which is enough abstract, flexible and powerful to be used on almost all compilation tasks. The framework contains several components, implemented over the DIR, that can be combined to build multiple language retargetable compilers. It supplies front-ends for several programming languages, back-ends for several computer architectures and a large number of code analysis and optimizations routines.

Except for some specific points, the DOLPHIN framework is not by itself an innovator project, but the away how we intend to use and the applications that we intend to give to this framework, are by themselves quite different and innovative comparing with all the other projects that we know on this area.

This paper proposes a full architecture, based on the DOLPHIN framework that integrates a large set of services related with compilers users, developers and teachers.

Keywords: Compilers development, frameworks

1 Introduction

DOLPHIN is a framework conceived to support the development of multi-language retargetable compilers. It is based on a solution implemented by Matos [4, 5], to test and develop compiler routines, later used to teach topics about compilers development. However, the absence of some essential tasks of the compilation process, took us to conceive a solution more complete and more adaptable to experiment new ideas. Initially, the goal was to build an application to implement, test and evaluate compilation routines that could be also used as a didactic tool. These goals could be satisfied developing simply a single compiler, used as a test-bed. However, we opted for a more ambitious solution that resulted on the DOLPHIN system and specifically on the DOLPHIN framework [10].

The main principle behind the conception of the DOLPHIN framework is that there are several advantages on the utilization of a common type of code representation along the compilation process. Of course, that this is not feasible or at least efficient for all compilation tasks, but the principle consists on use, whenever is possible, the same code representation. Why? Well, a uniform type of representation allows an easy reutilization of the routines, which is fundamental for the conception and development of a framework that aims to reduce the implementation costs, namely time and experience. Notice that the utilization of a common type of code representation is, normally, feasible only on the tasks that are not dependent of the characteristics of the source language (the ones used to write the code submitted to the compilation), neither of the characteristics of the computer architecture (the combination of the system operator and microprocessor over which the code generated by the compiler should run). DOLPHIN framework was conceived using this principle and contains essentially three types of components: the front-ends, responsible for the interpretation of the source language into the DIR; the back-ends, that pick the DIR and generate the output code, like: binary code, assembly or other higher forms of code representation (C, XML, etc); and the middle-level components, that work over the DIR, producing code transformations or obtaining data essential to the

execution of other tasks. On the middle-level, are included practically all code optimizations and analysis that are independent of the source language or the computer architecture.

Notice that our conception of a back-end is wide-ranging since a back-end can be a component that generates assembly or binary code, but also a component that generates information about the intermediate levels of the compilation process (normally produced as XML). So, in our conception a compiler could have several back-ends, some really at the end of the compilation process, but others in the intermediate stages.

The framework contains a fourth type of component, used to built the DIR. But these components belong to a different level, they are not used directly to build compilers, instead they are used implicitly by the other type of components. Notice that they should be available in the framework to allow the implementation of new components by other users.

The idea of supply all the necessary conditions to allow the implementation of new components by outsiders is another principle behind the conception of the DOLPHIN framework, and generally of the DOLPHIN system, and probably the most important principle. Why? Well, since the beginning of the conception of the DOLPHIN project that we understood that this project will require resources that we do not have, namely knowledge and workmanship. Surprisingly, this was not an obstacle, instead led us to plan a much more ambitious project requiring even more resources. Are you confused? Let explain our idea. Comparing to our framework, the most important reference is the SUIF compiler system [1,2], which is a project developed by the Stanford University in collaboration with many other universities and laboratories (University of Harvard, University of Toronto, University of Berkeley, etc), and with a substantial financial support of the Darpa and NSF. So, we had two solutions: join them or give-up. The last one was not an option for us, so we started to work on the first one and soon we understood that the SUIF compiler system is really a big big project and in many ways is exactly what we are trying to implement: a full system that can be used as a test-bed to put in practice our ideas, to teach topics about compilers and to build real compilers. More than that, it is freely available and has many people working on it that eventually could supply some support. Nice talk, isn't it? But you are even more confused now, aren't you? We always suspect that our project has many things in common with the SUIF compiler system, since they share most of the goals. The experience with the SUIF allows us to confirm this idea (after a considerable time spent to put the SUIF system to work and even more time to learn how to use it), but also to confirm that there are some important differences between the two projects and that are some things that could be improved in our project. This reinforces our determination to continue the implementation of the DOLPHIN system. But as already was told this could only be done with the required resources. It was when we start to think that our effort should be concentrated, at least initially, on put the DOLPHIN framework available and easily accessible to all potential users, expecting to get, directly or indirectly, collaborators to our project. It is way we established as a principle: supply all the necessary conditions to promote the implementation of new components by outsiders.

Of course, that for us is not sufficient to wait until the voluntaries appear or that somebody picks the framework only for personal purposes. By the other side, we do not want to restrict the access or the utilization of the framework. We believe that we can get more contributions if we promote the necessity of new components and if we supply all the necessary conditions, for the collaborators develop these components "inside" the system. We believe that this can be achieved if the solution is accessible, easy to use and satisfies all the essential necessities of the users, specially these ones that have knowledge about compilers development.

Taking into account all these conditions, we redefined our plans resulting on the actual state of the DOLPHIN. The framework and DIR still the central part of the system, but now there are several other sub-projects under plan based on the framework that aim to supply a set of services to the compilers users and implementers community. The accessibility is assured supplying the services via web; to turn the utilization of the several services less difficult, they are organized and supplied over distinct interfaces, one for each type of user; and to promote a controlled development of the framework, the implemented solution is centralized and offers an integrated development environment specifically conceived for the development of the framework.

The services and implicitly the sub-projects of the DOLPHIN, are organized taking into account the type of users, whose the most important for us is the compilers developer, for which is supplied the DOLPHIN – Compilers Development System (DOLPHIN - CDS). For the users that intend to develop new components for the framework, it is supplied the DOLPHIN –Compiler Components Development System (DOLPHIN-CCDS). It is also planned a more complete service, the DOLPHIN – COMPilers LABoratory (DOLPHIN-COMPLAB), which aims, by one side, an assisted development of new components, allowing the use of the framework for pedagogical purposes and, by the other side, the development of more complex components.

To promote the divulgation and utilization of the DOLPHIN, is also planned an Integrated Development Environment (IDE), available via Web for software developers, designated by DOLPHIN – Web IDE (WIDE),

which should be the show-case of the solutions implemented on the DOLPHIN framework. The WIDE should be based on a multi-language retargetable compiler with access to all code optimizations implemented on the framework.

Now that the main sub-projects were introduced, it is time to give more details. The next section explores the DIR and the framework; section three shows the type of functionalities that are planned for each interface; section four introduces the DOLPHIN – INNovation; section five explains the relation between the DOLPHIN project and the subject of this PhD thesis; and section six presents the conclusion.

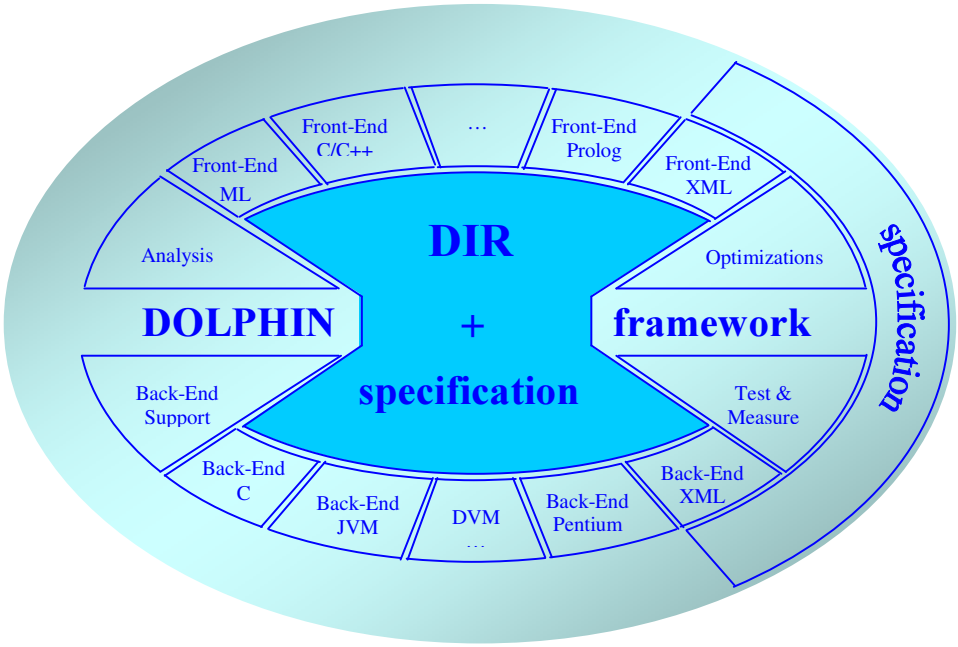

Figure 1 – DOLPHIN framework.

2 DIR and the DOLPHIN framework

The conception and implementation of the DIR is the central project of DOLPHIN system, which intends to specify the code representation model, the several interfaces supplied to the users and the correspondent implementation [3,4,5]. DIR is an object oriented code representation, which means that the code representation is done with objects. These objects are implemented as C++ classes, organized in four groups: the *FlowNode*, the *Expression*, the *DataTransfer* and the Generic.

FlowNode group contains all the necessary classes to represent the control flow operations of the code. These classes allow to build the Control Flow Graph of the program. The *Expression* group contains the classes used to represent the operations (including the control flow operations). The *DataTransfer* classes are used to characterize the data dependencies of the program and are closely related with the *Expression* classes, since they appear always associated. Finally, the *Generic* group contains all the other classes used on the representation of auxiliary, but pertinent information, such as identifier table, the modules of the program (blocks of code, procedures, functions, etc), and the self program as unique entity. Notice that the DIR classes are syntactically independent, which means that the way how the classes are combined to build the code representation is not subjected to any pre-defined organization.

The DIR classes supply several interfaces, allowing different modes to construct the code representation. At the most powerful, and also complex, interface the user must accomplish all the operations necessary to build the code representation. More elaborated interfaces can automatically execute some of these operations release the user of some details about the construction of the code representation and also about the DIR. For example, one of these interfaces avoids the utilization of the *DataTransfer* objects by the users; one other interface allows to build the code representation using only objects of type *Generic* and *Expression*; and there is one that only uses the *Expression*

classes. All these interfaces are accessed using C++, which introduces some unnecessary complexity. It is way we have also plans to build a script language to access the DIR objects, avoiding the details of the C++ language.

Another goal of this sub-project is the specification of the DIR, which will be done in two distinct levels: one that defines the classes as standalone elements, which includes the prototype of the methods and fields and, some semantic information about the behavior and the application of the classes; and a second level that defines the several interfaces available to build the intermediate code representation. These specifications will be part of the full specification of the DOLPHIN framework, which would be used by most of the applications that access the framework.

Specially, for the front-ends and back-ends, the specification should consider some extra information, such as: the syntax and helps of the source languages (for the front-ends), and the computer architecture specification (for the back-ends). This information will be very useful for some of the projects presented at the next sections.

The DOLPHIN framework comprehends the DIR, a specification of the framework and the components that will be used to build the compilers or, to test and measure the compilers performance. These components are organized in five groups: front-ends, back-ends, middle-level analysis, middle-level optimizations, back-ends support and measure/test components. Of course that this is the kind of work that never ends, there is always a code optimization to implement or a new language that requires a front-end. So, our idea is to implement only the necessary to put the system to work. Figure 1 shows the architecture of the DOLPHIN framework.

3 DOLPHIN services

As already was told the DOLPHIN project intends to supply four levels of services: one designed for the software developer (DOLPHIN-WIDE); one for the compilers developer (DOLPHIN-CDS); one for the DOLPHIN framework developers (DOLPHIN-CCDS); and one more for pedagogical purposes (DOLPHIN-COMPLAB). At this section is made a brief description of the type of functionalities that should be supplied by each of these services.

3.1 DOLPHIN – Web Integrated Development Environment

Technologically, the WIDE is the less important project, but it is probably the one that will attract more users, since intends to supply a very versatile IDE, that should provide all the essential things necessary for the software development, such as: software projects management and an attractive editor system (highlight with a help system), but also a debugger and, eventually, several simulators. The debugger should work for all compilers implemented based on the DOLPHIN framework, since the debugging should be executed over the DIR. The simulators are conceived simultaneously with the back-ends of the framework. The project DOLPHIN-CCDS consider the development of a simulation environment generator, based on the extra information supplied by the specification of the back-ends (of the computer architecture).

But the main part will be the compiler that should be constructed using the solutions implemented on the DOLPHIN framework. We intend to take advantage of the fact that the compilers built with the DOLPHIN framework share the same intermediate code representation, to supply a multi-language retargetable compiler, allowing to compile source code from different languages and to generate code for several computer architectures.

But WIDE is not only a bait to attract users or a show-case of the technology implemented on the DOLPHIN framework. We intend to use this project as support for one other project, designated by DOLPHIN – Integrate Development Environment Generator (IDEG), that makes the bridge among the CDS and the WIDE. The goal is to use the compiler specification implemented on the CDS and some information provided by the users to produce automatically a full IDE.

3.2 DOLPHIN – Compilers Development System

The DOLPHIN – Compilers Development System is our first big challenge. The main idea is to have an IDE over which the users will specify the architecture of the compiler, chosen the components and the order by which they should execute. To satisfy this goal, we intend to: supply a browser to navigate over the framework, accessing the information about the components, such a: goals, dependencies, precautions, etc; and two editors, one textual and one graphical. These editors should allow to specify the structure of the compiler and parameterize the components . Notice that the implementation of the textual editor implies the design of a language to specify the structure of the compilers.

The IDE will be supported by a generator that picks the specification (written on the textual or graphical editor), to automatically build and download the compiler. To normalize the operations of access and manage of the framework, there is another project, designated by DOLPHIN – Framework Management System (FMS), that has two goals: the design of protocol to manage the framework (to access, update, remove or insert components); and the

construction of an application to implement the protocol and to make the interface among the framework and the several types of user.

As already was told, we intend to provide means for the user automatically generate an IDE for the compiler, (DOLPHIN-IDEG). This is feasible using the "static" features of the WIDE, such as: the debugger and the IDE helps; and some specific features that are dependent of the compiler architecture. For example: the specification of the extra information of the front-end should allow to generate the highlight editor and the helps for the compiler source language; the simulator should be provided as an add-in of the back-end (they should be built at the same time). Even the compiler specification will be used to generate the selection box where the compiler users can select the code optimizations that should be applied during the compilation.

3.3 DOLPHIN – Compiler Components Development System

DOLPHIN – CCDS, was already told, aims the development of new components for the DOLPHIN framework. It is a fundamental project in our strategy, since the idea is the construction of "normalized" components. We reinforce the words "normalized components". Why? For us, are the type of components that are very useful and, as consequence, important for the affirmation of our project, but who's implementation, by one side, does not require strong knowledge's about the DOLPHIN framework or about compilers implementation and, by the other side, can be built using appropriate tools (normally based on a specification and on a generation system). In other words, the components that can be built by outsiders, that do not want to lose too many time or that do not have deep knowledge's to implement them "by hand" (without the supporting tools). For the development of more complex components, it is possible to use the DOLPHIN-COMPLAB.

But the intention of supply tools to help the development of new components makes the DOLPHIN-CCDS the most ambitious project of the DOLPHIN system. The tools are organized in three groups: the ones used to support the construction of front-ends (generators of the lexical, syntactic and semantic analyzers + highlighter editor + source language helps systems); the ones used to support the construction of the back-ends (generator of the instructions selector, registers allocator, low level code optimizers and output code generators + simulator/simulation environments generator) [4]; and the tools for the construction of the middle-level analysis and optimizations components (frameworks for Data Flow Analysis, Control Flow Analysis and Code optimizations + generator of code rewrite routines) [13, 14]. Practically, each of these tools is a big and very hard to implement project, so we know that it is not feasible for us to implement all these tools in acceptable time. But we expect that, if our strategy to get users for the DOLPHIN framework, works that some contributions appear for the development of this part of the DOLPHIN system. Anyway, there are some prototypes for some of these tools, like a back-end generator (DOLPHIN-BEG) [4], and two frameworks: one for Data Flow Analysis and other for Control Flow Analysis.

3.4 DOLPHIN – COMPilers LABoratory

DOLPHIN – COMPLAB is the zenith of this project, since the goal is to implement a full laboratory for compilers development, that can extend the capabilities of the DOLPHIN-CCDS allowing the development of more complex components, but that work specially as an application to teach topics about compilers development and, as a test-bed for the implementation and valuation of new solutions [12]. We believe that these goals can be achieved given a free and complete access to the DOLPHIN framework, supported by the integration of the several services supplied by the DOLPHIN system.

Notice that the DOLPHIN-COMPLAB should support the full development of new components, which comprehends the implementation, the test and the valuation of the components, which includes the performance measure. The implementation could be supported using the DOLPHIN-CCDS and some tools for Components Direct Development (CDD). A realistic test and valuation implies the construction of a compiler (or even more than one compiler) that uses the component under development. This can be done using automatically the DOLPHIN-CDS and a DOLPHIN-WIDE specifically generated for tests (using the DOLPHIN-IDEG). But this special DOLPHIN-WIDE should supply some extra mechanism, such as benchmark tests for the compiler (and indirectly for the component under construction). Notice that this is a hard task since requires assemble test routines for several programming languages. Another goal is to supply a more advanced debugging system based on the graphical representation of the DIR, designated by DOLPHIN-Monitor and Analyzer System (DOLPHIN-MAS) [7,9,11].

Even the compiler generated to test the component under development should contain some special features, especially to measure the performance of the compiler, that will be implemented as components of the framework.

But to put this to work it is necessary a management system to deal with the framework. Considering the two main purposes of the DOLPHIN-COMPLAB (development of the framework and teach compilers development topics), we intend to build an IDE over which is possible to create projects. Each project is based on an instance of

the framework and accepts two levels of users: project coordinators and conventional users. The project coordinator is responsible for the creation of the project and is the one that makes the management of the conventional users, creating accounts and setting the permissions to access the components and tools. It is also the type of users that would have access to submit components to be integrated on the original framework. We still investigate new ideas to supply other services that could be useful for the coordinators, especially for teachers.

Conventional users are the ones that effectively make the development of the framework. They will have access to practically all services supplied by the DOLPHIN system (CCDS, CDS, WIDE, IDEG, etc). Notice that only the instantiated framework will be shared. Compilers specification, components not yet integrated on the framework, environment preferences, and other personal items will be accessed only by the owner.

Probably, the IDE will supply a message management system, a dash board, a log-book, a task management system, etc. Figure 2 shows the schematic representation of the DOLPHIN system.

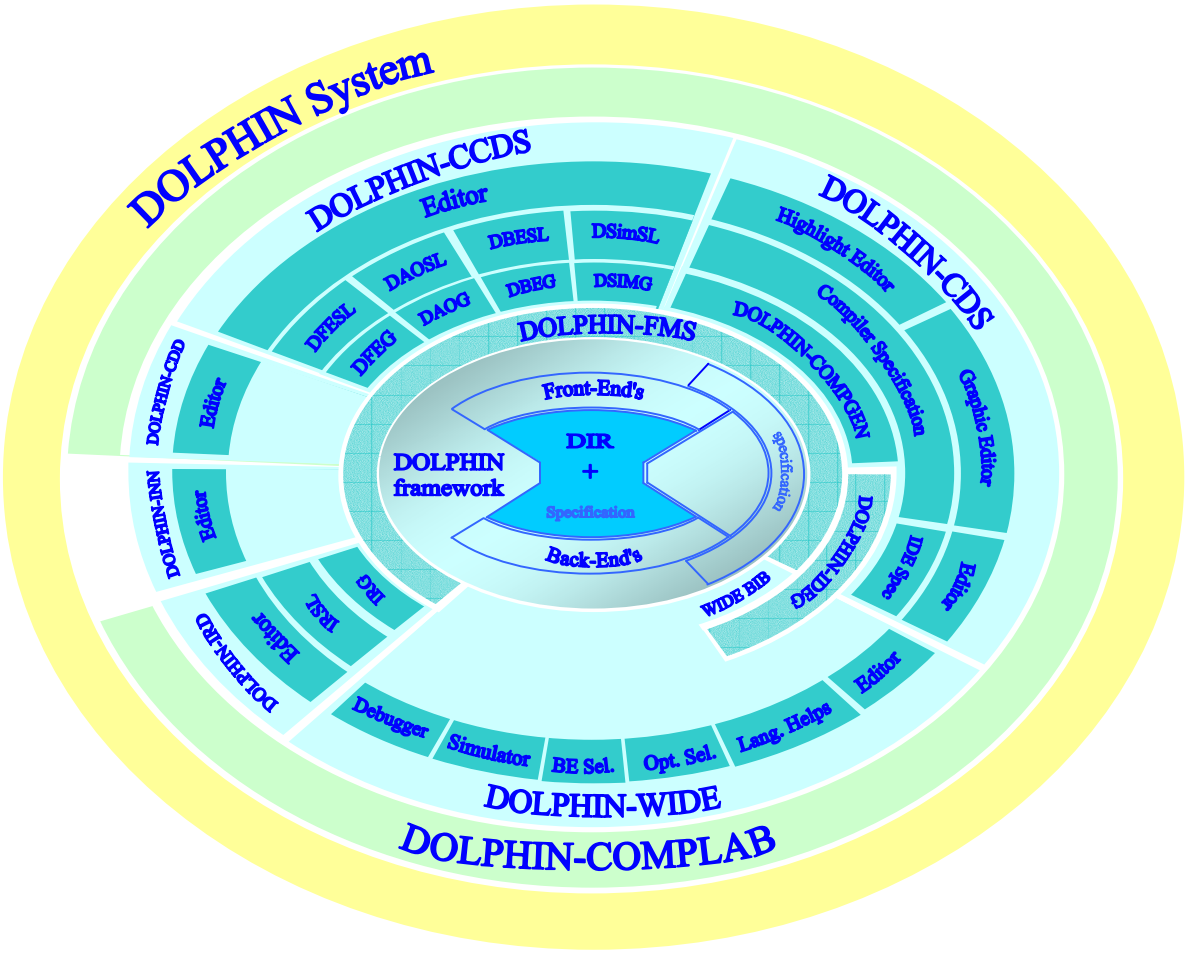

Figure 2 – Part of the DOLPHIN System.

4 Other projects of DOLPHIN system

Two other projects that deserve a reference, are the DOLPHIN- Intermediate code Representation Definition (DOLPHIN-IRD), and the DOLPHIN – INNovation (DOLPHIN – INN).

For us, DIR is a quite flexible and powerful form of code representation, but we have conscious that DIR is not prepared to represent all types of programs. For example, DIR have not been tested on the representation of concurrent programs. By other side, one of the ideas that we had, was to test the viability of use the DIR on the representation of problems from other areas [6, 8]. Notice that this would allow to use, without deep modifications, the solutions implemented on the framework to solve or help the resolution of these problems, for example: use data

flow analysis techniques for decision support systems; or the scheduling and code parallelization techniques to solve planning problems. The attempts that were done exposed the limitations of the DIR, namely to deal with undeterministic problems (with a strong statistic component), or with highly dynamic systems.

All these reasons led us to plan the DOLPHIN – IRD, which aims the implementation of the necessary features to help the modification and expansion of the DIR We do not know exactly how to do this or if is feasible, but we believe that would be an advantage for the DOLPHIN system.

The other project, DOLPHIN-INNovation intends to test the possibility of use some artificial intelligence techniques to solve some tasks of the compilation process. This is a very recent idea that tries to use a totally different approach. The solutions used nowadays, are deterministic and static, they are not able to adapt themselves to the details of the source language or the computer architecture, neither improve the performance (generate better code and/or generate the code faster). The goal of the DOLPHIN-INN is to apply agents with neural networks to solve classical compilation problems, namely register allocation, instruction selection and even code optimizations.

5 My dissertation

Well, now that the DOLPHIN system was briefly introduced, it is time to explain its relation with my PhD work. What I am trying to prove is that the DOLPHIN system and more generically the ideology behind the DOLPHIN is realistic and feasible, which will be done proving that the next goals are practicable:

1. Show that DOLPHIN system is feasible and offer several advantages to promote and implement new compiler solutions;
2. Show that DOLPHIN system has several advantages to teach topics about compilers development (probably is the unique solution that supplies this type of features);
3. Show that the DIR and more generically the DOLPHIN framework have lots of potential, not only at the point of view of compilers development, but also as a generic system to solve or help the resolution of problems from other scientific domains.

Some parts of the DOLPHIN system were already implemented, for example: DIR is implemented and is quite stabilized; the framework contains enough components to be used, namely code analysis and optimizations routines; the DOLPHIN-CDS is under implementation and can already be used; there is a prototype of the DOLPHIN-WIDE (without the simulators, debugger and helps), a prototype for a back-end generator and a prototype of a DFA framework; and even the DOLPHIN-MAS is partially implemented.

Things that I intend to implement until the end of this PhD work, are: the specification of the DIR and of the framework; conclude the DOLPHIN-CDS; implement the DOLPHIN-FMS; implement a prototype of the DOLPHIN-IDEG; implement a more complete prototype of the DOLPHIN-WIDE; build the IDE of the DOLPHIN-CCDS and of the DOLPHIN-COMPLAB. At the end, measure the usability and utility of the system.

6 Conclusion

Even without be fully implemented, it is already possible to see the potential of the DOLPHIN. Any project built to help the development of compilers or even compiler components, is certainly a project that contains many challenges. But DOLPHIN is much more than that or at least intends to be, since does not limit its goals to the simple support of compilers development, it intends to do it well and efficiently, but also accessible for several types of user: software programmers, compilers developers, teachers and students. In our opinion, all these goals were or will be achieved, since DOLPHIN system:

1. Supplies a framework that is accessible, open, functional and that contains already a considerable number of components;
2. Allows and promote the implementation of new components for the framework, supplying tools and environments specially conceived for this purpose;
3. Supplies support for the investigation and teaching of compiler;
4. All the services are or will be available via Web;
5. Intends to investigate non conventional solutions.

We also hope to obtain very good results on the application of the DOLPHIN framework to represent, optimize and simulate solutions for problems from other scientific domains, namely on: production management, planning and decision support. It is also perceptible that DOLPHIN still have space to grow and certainly, will appear many new goals. Some results are already available via Web, but in our opinion, only achieving all the defined goals, it will be possible to show with exactness the advantages of the DOLPHIN system.

References

[1] Aigner, G., Diwan, A., Heine, D., Lam, M., Moore, D., Murphy, B. and Sapuntzakis, C. The SUIF program representation. Stanford University, Computer System Laboratory, 2000.

[2] Aigner, G., Diwan, A., Heine, D., Lam, M., Moore, D., Murphy, B. and Sapuntzakis, C. The basic SUIF programming guide, Stanford University, Computer System Laboratory, 2000.

[3] Johnson, R.E., McConnell, C. and Lake, J.M. The RTL System: A framework for code optimization. Proceedings of the International Workshop on Code Generation, Dagstuhl, Germany, pp. 255-274, 1991.

[4] Matos, P. Estudo e desenvolvimento de sistemas de geração de back-ends do processo de compilação. Master thesis, Universidade do Minho, Braga, Portugal, July, 1999.

[5] Matos, P. and Henriques, P. DOLPHIN framework. Technical report, Universidade do Minho, Braga, Portugal, October, 2002.

[6] Matos, P. and Henriques, P. Data Flow Analysis applied to optimize generic workflow problems. Proceedings of the International Conference on Industrial Engineering and Production Management, Porto, Portugal, May, 2003.

[7] Matos, P. and Henriques, P. DOLPHIN-FEW - An example of a Web system to analyze and study compilers behavior. Proceedings of the International Conference e-Society, Lisbon, Portugal, June, 2003.

[8] Matos, P. and Henriques, P. Applying compilers technology to solve generic workflow problems. Actas do 3º Congresso Luso-Moçambicano de Engenharias, Maputo, Moçambique, August, 2003.

[9] Matos, P. and Henriques, P. A solution to dynamically build an interactive visualization system to the DOLPHIN-FEW. Proceedings of the International Conference on Visualization, Imaging, and Image Processing, Spain, September, 2003.

[10] Matos, P. and Henriques, P. DOLPHIN-FEW - An architecture for compilers development, monitoring and use on the Web. Proceedings of the Fifth International Conference on Information Integration and Web-based Applications and Services, Jakarta, Indoneisa, September, 2003.

[11] Matos, P. and Henriques, P. Construção dinâmica de um sistema interactivo para visualização do código intermédio do processo de compilação. Actas do 12 Encontro Português de Computação Gráfica, Porto, Portugal, October, 2003.

[12] Matos, P. and Henriques, P. DOLPHIN-COMPLAB: A virtual compilers laboratory. Proceedings of the Second International Conference on Multimedia and ICTs in Education, Badajoz, Spain, December, 2003.

[13] Nielson, F., Nielson, H. and Hankin, C. Principles of Program Analysis. Chapter 2, Springer Verlag; ISBN: 3540654100, 1999.

[14] Tjiang, S. Automatic Generation of Data--flow Analyzers: A tool for building optimizers. PhD thesis, Stanford University, Computer Systems Laboratory, 1993.