Abstract

DOLPHIN is a framework for the development of high performance, multi-language and multi-architecture compilers. This framework contains several components, but is specially prepared to help the development of the middle-level task of the compilation process. It contains a large set of analysis and optimizations routines, several front-ends and back-ends, and a set of tools to help the development of new routines. This paper focus the DOLPHIN – Front-End for the Web, a project based on the DOLPHIN framework, to build a set of applications, to make available some new services on the web, namely an integrated development environment, a system for monitoring and analyze the behavior of the compilation tasks and also, an environment for the development of compiler components.

Keys words: Compilers development, frameworks, web applications

1 Introduction

DOLPHIN is a project that intends to build a framework, also designated by DOLPHIN, to support the development and study of high performance, multi-language and multi-architecture compilers. This project started as a test-bed to implement and test compilers routines. Later was used as a tool to teach topics about compilers development. But the necessity of a full environment, that contains all essential compilation tasks, took us to project a new solution.

Initially, were established three main goals: build a full environment to test, implement and measure new solutions; build a pedagogical tool to teach topics about compilers development; and build a tool to help the development of new compilers [7, 8].

Looking to the Figure 1, it is possible to see the architecture of a conventional compiler. The initial tasks are related with the interpretation of the source language, and together they form the front-end of the compiler (lexical, syntactic and semantic analysis). The last tasks are related with the output code generation and form the back-end of the compiler (registers allocation, instructions selection, assembly/binary code generation). There are also some tasks that can be executed along the compilation process, such as the code optimizations, some code analysis and error detection.

Most of the established goals imply the implementation of several front and back-ends. To save time and increase the code portability, we decide to design a code representation form, to use between the front and the back-end, typically called by middle-level of the compilation process. We want a code representation, independent of the source language and the computer architecture characteristics. This, by one side, guarantees the portability of the routines implemented over it, and, by the other side, promotes the implementation of new routines at this level of the compilation process.
It was at this point that appear the idea of a framework, composed by a set of routines, that work over the designed form of representation and, a set of front and back-ends, that can be easily parameterized to build new compilers or experiment new solutions. It was when the DOLPHIN framework born, since then many new ideas have appeared around the DOLPHIN project. One of most successful and original is the DOLPHIN – Front-End for the Web (DOLPHIN-FEW), a sub project of the DOLPHIN that includes all components that are related with the web.

The DOLPHIN-FEW architecture is also the main subject of this paper that is divided in five sections: one with the introduction (this one); one describing the code representation form; a third, showing DOLPHIN architecture and the mechanisms available for the development of new components; a fourth section, that describes the goals, components and future plans of the DOLPHIN-FEW; and a last one with the conclusion.

2 DOLPHIN Internal Representation

The classical forms of representation are based on expressions trees or the list of triples (or quadruples), composed by the operator and operands (eventually, the result of the operation). The beautiful of these solutions is the simplicity; they are easy to represent, easy to work, easy to understand and, probably, are the best solution if we plan to build a compiler with a classical architecture.

But we think on the DOLPHIN framework as a much more ambitious work. First, it is not a compiler, at least is not a simple compiler. Even, restricting DOLPHIN to the DIR, to the implemented routines and, to the front and back-ends, we still have a powerful compiler that has multi-language and multi-architecture support, and a strong optimization component. But the DOLPHIN framework intends to be much more than that. For example, for us, the code representation should not be restricted to represent only the source code. There is much more information that is produced along the compilation process that is also part of the representation, since it allows a better comprehension of the program.
Considering, the goals defined for DOLPHIN, we conceived a more elaborated form of representation. Our solution, here designated by DOLPHIN Internal Representation (DIR), is based on the one used at the RTL System [6]. The code is represented using a set of C++ classes that can be classified in four categories: FlowNode, Expression, DataTransfer and Generic. The first category contains the classes that are used to represent the control flow of the source code, building the Control Flow Graph (CFG). The Expression classes represent the expressions trees. The DataTransfer classes allow a linear representation of the code, but are used specially to control the data dependencies between expressions. The Generic category contains all the other classes that can not be included on the other categories, such as Program class, Function class, IdentifierTable class and many others. Figure 2 shows the principal classes contained on each of these categories.

Figure 2 – Hierarchy of some DIR classes.

Notice that DIR objects do not follow a rigid structured; instead, they form a set that can be combined to build the code representation, but the way how this is done depends on the user (the routine that builds the intermediate code representation).

Some people may think that the DIR is much more complex than a classical solution, but this is not necessarily true. To build the code representation the user has to work essentially with the Expression classes, which is very similar to the classical solution that uses the expressions trees. The objects in DIR take care of everything, building automatically the complete representation that contains much more and, better information, than the classical solutions. For example, the data dependencies and the type analysis are partially executed while the code representation is built. More complex information, like the Control Flow Tree or the Program Dependence Graph, is computed immediately after the construction of the representation and, if occurs any transformation on the code, the computed information is automatically actualized.

DOLPHIN is not intended to be a compiler, but a framework to develop compiler components or, eventually, complete compilers. To satisfy this goal, it is important to use a flexible form of representation that can be easily adapted to support other information or other operations. This can be easily done on the DIR, the user only has to derive new classes from the existent ones, to add new variables or methods.

But these are not the only reasons why we choose this form of representation. We have many plans for the DOLPHIN framework, one of them is to build a set of tools (some are already under construction) to help the development of new solutions, particularly for some code analysis and optimization routines. This is an ambitious goal, but the intention is to go much more ahead. It is under development, a new language that can be classified as declarative, and that is able to access the DIR objects (and even the objects that implement the routines). The objective is to use this
language to describe the effects of the analysis and the optimizations routines and, then let the tools generate, full or partially, the correspondent routines and data structures, without worry about with the implementation details. Even without results, we believe that the DIR is specially adapted for this job.

There is one more reason to choose this form of representation. We are trying to adapt some technology, used for compiler construction, to problems from other areas, namely at planning and process management. To test the solutions for this type of problems, is crucial to have a more abstract form of representation, that can be easily adapted to represent the elements of these problems. This is possible with a solution like the DIR but very difficult with a classical solution. Of course there are some disadvantages, for example, the DIR requires more computer resources (memory) than a conventional solution, but this is the price to pay to have pre-computed information. The retribution is obtained at the compilation time and at the type of solutions that can be implemented.

3 DOLPHIN architecture

The architecture of DOLPHIN was conceived, considering our conceptual idea of the DOLPHIN framework and, indirectly, the defined goals. The principles are:

- First, if a routine is independent of the characteristics of the source language or of the computer architecture, then it should be implemented at the middle-level, directly over the DIR.
- Second, if a routine do not satisfy the condition of the first principle, then it is supposed to be implemented on the front or the back-end (depends if is dependent of the source language or the computer architecture).
- Third, the function of the front-end is the interpretation and the conversion of the source language to the DIR. It should not execute any kind of code optimization, giving more chances to the middle-level routines produce better results.
- Fourth, always that possible, should be done an effort to move the routines, that are typically executed on the front or the back-end, to the middle-level, increasing the portability and reutilization of the routines. Sometimes, this is done simply by moving the routines to the middle-level; but many times, it is required a parameterization, in function of the characteristics of the source language or the computer architecture.
- Fifth, it should be done an effort at the middle-level to provide all required support to the back-end routines.

Taking in to account these principles, it is natural to consider the DIR and the solutions implemented at the middle-level, the soul of DOLPHIN. Figure 3 represents the complete architecture of DOLPHIN, including DOLPHIN-FEW. It is possible to see the DIR in the center of the representation. Over it, are implemented the middle-level routines, the ones that belong to the DOLPHIN framework and others implemented by the users. These routines are specially devoted to the code analysis, collecting information about the program that is later used by other routines, namely by the code optimizations, that operate transformations on the code to reduce the execution time, the required resources or, more generically, to improve the quality of the output code.

At present, DOLPHIN contains several solutions for the control flow analysis (CFA) and distinct types of data flow analysis (DFA), like the Reaching Definition Analysis, Live Variables Analysis, Available Expressions Analysis, even some experiences with Alias Analysis based on DFA. The data dependency analysis is supported directly by the DIR and by some other routines, like the one that builds the Program Dependence Graph (PDG). There is also a large set of code optimizations and, all the required mechanisms to convert the DIR to and from the Static Single Assignment form (SSA) [4, 2]. Almost all analyses and optimizations routines can work on the normal and on the SSA form.
The routines that give Support to the Back-End Tasks (SBET) also work over the DIR. Two examples are: the routines that build the PDG and the Data Dependence Graph, to support the instruction scheduling optimizations.

Around the DIR and the middle-level routines, work the several front-ends and back-ends. There is already implemented a prototype for C language front-end and, recently, started to be implemented a front-end for C++ language. There is also two other experiences, one with a Prolog front-end, derived from another project that uses the framework, to build a compiler to robot programming language [1]; and other experience, to design a language to describe generic workflow problems (Program Specification Language). But the idea is to implement many more front-ends, namely for other paradigms, for example, to functional languages like ML.

DOLPHIN also contains a prototype of a back-end to generate code to a Pentium processor (using the essential of the instruction set), a prototype to generate C language (yes C), and exist some experiences with the Java Virtual Machine. But the objective is to implement many more back-ends, namely to the Common Runtime Language of the .NET framework.

3.1 The development process

One of the goals of the DOLPHIN framework was the development of a solution to build new compilers or new compiler components. Looking again to the Figure 3, it is possible to identify a subproject designated by DOLPHIN – Compiler Components Development System (DOLPHIN-CCDS). This subproject integrates several tools (also designated by components), built to help the development of new compiler components (routines), particularly for the middle-level and backend.

One of the components of DOLPHIN-CCDS is the DOLPHIN Analysis and Optimizations Generator (DAOG), a subproject in development that aims to build a tool to generate code analysis and optimizations routines. The main idea is to design a language, the DOLPHIN Analysis and Optimizations Specification Language (DAOSL), to specify the analysis and optimizations. The specification is then submitted to the DAOG to generate all the necessary data structures and routines. The DAOSL has been designed as a declarative language, able to work with objects (not...
necessarily object-oriented). These objects represent the elements of the code representation (objects of the DIR), but also routines.

The DAOG is implemented, based on distinct solutions. Probably, the simplest one, consist on a set of pre-implemented routines, parameterized with the information provided by the specification. This is applied, specially, at some back-end routines, for example, the variables interference graph (VIG), used on the global register allocation routines [3], is the type of data, typically, associated with the back-end tasks, since requires information about the processor registers. But the VIG can be more easily built at the middle-level, using the results provided by the Live Variables Analysis (a form of Data Flow Analysis). The only problem is the lack of the information about the registers. DAOG can generate the routines to build the VIG based on a specification of the processor instruction set.

DAOG has also a prototype to generate DFA routines. It is based on the framework proposed by Flemming Nielson et al. [10], where the user specifies: the data structure that should be used (designated by lattice); the initial value; the join operator, that defines how the values are combined (when n data flow paths converge to a single one); and a set of transfer functions, that describe the effects of the several operators over the lattice. The computed values are later used by other routines. This solution is also used to expand or to change the native DFA routines, implemented at the DOLPHIN framework.

Unfortunately, DAOG still not a solution that can consistently work the results obtained with the DFA routines. The main problems are derived from the design of the DAOSL that does not satisfy our pretensions and, from the lack of a framework, or a parametric solution, that can be used to generate the optimizations routines (the ones that typically deal with the results of the DFA routines).

But DAOG has also another type of solution, for example, it is possible to describe some code optimizations (probably, a better name is code transformations), using a pattern matching and rewrite techniques. The specification defines a set of rules, each one contains two expression patterns, one for the expression that is replaced and the other for the new expression. The routines generated by DAOG process, individually, each expression tree (in practice they are direct acyclic graphs), applying consecutively the pattern recognition routine until no changes are produced on the expression tree.

Another component of DOLPHIN-CCDS, is the DBEG – DOLPHIN Back-End Generator. It is a tool to generate the instructions selector and the local register allocation routines. The solution is based on the IBURG [5, 12, 13], that also uses a tree pattern matching and rewrite techniques [11]. But DBEG was modified to compute, at run time, the cost of each pattern and, to accept context conditions, which watch if is, or is not, possible to apply the pattern.

DBEG can also generates the register allocation routines and integrate them with the instructions selector routines. To specify the pattern matching rules and the register set, it is used the DCASL – DOLPHIN Computer Architecture Specification Language.

It is also possible to develop routines without using the DAOG or the DBEG, which is called by Components Direct Development (CDD). But all the three mechanisms use the DOLPHIN Component Management System (DCMS), which makes the interface between the development tools and the DIR, establishing the order by which the several middle-level routines should be called and, eventually, to insert some operations that must be executed between each call (for example, to remove, clean or actualize the data structures or to generate some monitoring information).

Presently, the instructions are passed to the DCMS using the implementation language of the DIR (C++), but it is planned the design of a meta-language, that makes the instructions more readable and transparent to the users, avoiding to deal directly with the objects (particularly with the methods) of the DIR.
Two other components, developed to make the interface with external components, are the XG and the DIRenv. The first, converts DIR to eXtended Markup Language (XML) and, the last one, converts XML to DIR. Both were developed as consequence of the subproject DOLPHIN-FEW (that is introduced at the next section), but they have been so successful, that we thought to use them on the self DOLPHIN-CCDS, to work as interface between the DIR and the DCMS. Potentially, it is now possible to implement routines to work directly over XML. It is also important to refer, that the development of these two components, took us to design a XML Schema to the DIR.

Another project that is only on the plan is the construction of a virtual machine (we do not know yet if belong to DOLPHIN-CCDS), that can run directly DIR. We designated it by DOLPHIN Virtual Machine (DVM). It is also planned a monitoring program to work over the DVM.

4 DOLPHIN-FEW architecture

DOLPHIN-FEW was born from the lack of an efficient solution, to observe and analyze the effects of the routines over the DIR. Notice that the conventional debugging does not satisfy our requirements, the compilation is a long process, with many tasks and many levels of representation (some very difficult for the human interpretation), and requires a careful observation. The first solution developed, to fulfill this lack, consist on the implementation of several routines to generate a set of text files with the information about the several stages of the compilation process. But derived of the quantity of information produced and the severe transformation of the code between the stages, the interpretation of the files content was a very hard task, sometimes impossible. We felt the necessity of a solution that provides mechanisms to navigate over the information. So, the next solution was to generate a set of files formatted with a hiper-text language that obviously was the HTML. This solution had several advantages: first, using the hiper-links it was very easy to relate the information; second, the files could be read and navigated using a simple web browser. It was quite a success, the analysis of the routines behavior, which was a tricky task, suddenly started to be very easy to do, allowing to:

- Navigate over the several code representation of the DIR (CFG, expressions list, etc);
- Relate distinct types of representations, for example: associate an expression with the correspondent edge of the CFG;
- Access the information associated with elements of the DIR, for example: attach variables that appear somewhere between the expressions with the correspondent information of the identifiers table;

But most important was the results that we did not expected, for example the solution allows to

- Relate elements of distinct stages of the compilation process, which is nowadays a fundamental mechanism to analyze the routines behavior;
- Visualize temporary information derived from the code analysis and optimization routines;
- Relate things that until here impossible to relate, for example: relate the memory addresses, that appear at the last stages of the compilation process, with the correspondent identifiers.

Figure 4 represents an example of the solution based on the HTML. It is possible to see that the example has many links with hexadecimal values (addresses of the objects applied to build the DIR). Without a solution that contextualizes and relates these values with the correspondent objects, it was very difficult to analyze the DIR.

Of course that the next step, was to put this solution available over the web. Building a web page to upload the file submitted to the compilation to a Common Gateway Interface (CGI). This one calls a compiler, built based on the DOLPHIN framework, to generate the HTML files, then replies with the main file.

The results were very good, opening a new door to the development of DOLPHIN, giving origin to the DOLPHIN – Front-End for the Web (DOLPHIN-FEW). This project contains all components
related with the web, including the development of the DOLPHIN web site (http://www.labinf.estig.ipb.pt/dolphin).

**Control Flow Graph**
- Node:0032B710
  - Type: Conditional Node
  - Output (true): 0032B9E0
  - Output (false): 0032BF78
- Var. definition-Expressions (BOTTOM)
  - a(0032B060)
- Data Transfer's of the node
- Expression's of the node

**List of Expression's**
- Expression's of the node 0032B710
  - Address: 0032B8A8
    - Type: Memory
    - Data Transfer: 0032B8F0
    - Variable: a
  - Address: 0032B798
    - Type: Conditional jump expression
    - Data Transfer: 0032B7E8
    - Conditional Expression: 0032B8A8
    - Jump (False Cond.): 0032BF78
    - Jump (True Cond.): 0032B9E0

Figure 4 – Control Flow Graph and the list of Expression’s represented at HTML format.

DOLPHIN-FEW is also the main subject of this paper. It is a rich project with many original ideas and with a set of ambitious goals. This section presents the goals, the components that are implemented and the ones that we expect to be implement, but particularly the architecture of DOLPHIN-FEW.

DOLPHIN-FEW not only reinforce the objectives of DOLPHIN, as also, accomplish some of these objectives, namely: provides a more efficient and attractive pedagogical tool and, the necessary mechanisms to analyze and monitoring the behavior of the routines. But DOLPHIN-FEW has its own objectives, for example, supplies an integrated development environment (IDE) available via web, that includes most of the advantages of the DOLPHIN-framework.

But probably the most innovator idea of DOLPHIN-FEW, is to build an IDE for the development of new components for the DOLPHIN framework, using all tools implemented at DOLPHIN-CCDS (DAOG, DBEG and DCMS).

### 4.1 DOLPHIN Web Integrated Development Environment

As a web project, the first developed component of DOLPHIN-FEW was an IDE, designated by Web IDE (WIDE), that was initially developed to improve the interface with the user (to submit the files to the CGI). But sooner, appears the idea to expand it to a full IDE, exposing the potential of DOLPHIN [9].

WIDE, as any other IDE, allows the development of software projects, including edition and compilation of the code files. But has also some particularities that make it unique, for example:

- Potentially, it can offer most of the technology available at DOLPHIN, namely: multi-language support, multi-architecture support and a large set of code optimization routines;
- It is an IDE available via Web, this means that can be used in any place that has a computer with access to the Internet;
- It works as an interface to set the DOLPHIN compilation options, namely to: the code optimizations, the target computer architecture, the information that the compiler should generate, etc.
- It allows the analysis and the observation of the solutions implemented at DOLPHIN, using the generated HTML files;
- As consequence, is a precious tool, easily accessible, to teach topics related with compilers development;
And a fundamental tool to supervise the development of new components for the DOLPHIN framework.

Figure 5 shows the picture of the first version of the WIDE. It is possible to access it over the DOLPHIN web site.

4.2 DOLPHIN Web Monitoring and Analysis System

The second component of DOLPHIN-FEW, was born from the HTML solution that was implemented to monitoring and analyze the routines behavior. This solution is efficient but very little attractive, particularly when applied as a pedagogical tool. Normally, the behavior analysis of the routines is a localized task, falling upon over specific forms of representation (CFG, list of expressions, etc). However, it is required a sporadic access to the remaining information, specially to the one that is transversal to the compilation process, such as the information about the variables. For example, the analysis of the list of expressions requires information about the variables that appear at the expressions. It is possible to access this information, jumping for the page of the identifiers table, but this solution displaces the user from the main context and, exposes many undesired information. It was patent the necessity of new mechanisms to improve the interface with the users.

The first attempt to solve this problem uses scripts, inside the HTML files, to animate the links, for example, every time the mouse rolls over a link to a variable, it is opened a pop-up window with the information associated to the variable. But the introduction of the script put several problems to the HTML file generation, since requires information that was not easily available. And the effects produced with script are limited, for example, is not easy to change the initial format of the HTML files (established at the generation time).
The aim of improve the interface, resulted on the second component of the DOLPIN-FEW, the Web Monitoring and Analysis System (WMAS). It is an interactive graphical interface, dynamically built from the information generated by the compiler. It uses a set of new technologies, like the eXtended Markup Language (XML), XML Schemas (here designated simply by Schemas) and, Macromedia Flash\(^1\) components (specially developed for this solution).

Figure 6 shows the architecture of solution based on the HTML files and the architecture of the new solution, implemented with XML and Flash components. At both architectures, the user submits, over the WIDE, the source file to the CGI. This one calls the compiler to generate the information files. However at the first case, the compiler generates HTML files and, the CGI replies with the main HTML file. At the WMAS, the compiler generates XML files and, the CGI replies with the Flash application, passing also the URL of the XML files. When the Flash application starts to execute on the browser, it makes the download of the XML files and, builds the graphical representation.

The development of the WMAS, force the implementation of new routines at the DIR, to generate the XML files. Notice that, now the important is not to pass the documents to the client, but the information. The client is responsible for the visual representation and interaction with the user.

Each object of the DIR results on a xml element, represented by the identifier of the self object. Each of these XML elements contains other elements, one for each variable of the object. These internal elements are represented using the same methodology, which is: each element is represented by the identifier of the correspondent variable and, the value of the element is the value of the variable. If the variables are composed (arrays, data structures, classes, etc), then the content of each one is a set of other XML elements.

To avoid some problems that occurred at the generation of the XML, it was implemented a XML Schema to validate the XML files. The design of the Schema, revealed an excellent option, since allowed the detection of several errors that occurred on the generation process (unclosed and cross tags, generation of redundant information, etc). And now, is possible to work over the XML,

\(^1\) Flash is a registered trademark of the Macromedia, Inc.
producing transformations on the XML content and, then validate the result, to see if everything is correct and compatible with the DIR. If so, it is possible to convert again the result to the DIR. The Schema design was done, representing each class using the ComplexType of the XML Schema. Figure 7 shows partially two DIR classes with an inheritance relation (FlowNode and JumpNode), the Schema for these classes and the XML of an object of type JumpNode.

```java
class FlowNode{
    TNode tnode;
    ...
};
class JumpNode : public FlowNode{
    Jump jp;
    ...
};
JumpNode jump;
```

```xml
<complexType name="FlowNode">
    <sequence>
        <element name="tnode" type="TNode"/>
    </sequence>
</complexType>
<complexType name="JumpNode">
    <complexContent>
        <extension base="FlowNode">
            <sequence>
                <element name="jp" type="Jump"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

After the generation of the XML, the CGI sends to the client browser the Flash application, passing the URL of the XML files. The Flash application makes the download of these files and read the content to build an object, designated here by XML object, using the DOM API (defined by the W3C). Next, creates an instance of the component DefaultObject to start building the visual representation. Figure 8 shows two DefaultObjects, one on the standalone format and the other on the pop-up format. There are several derivations of the DefaultObject, specially prepared to represent specific forms of data structures, such as lists and dictionaries. It is important to refer that the DefaultObject can be used to represent any kind of XML object.

![Figure 8 - DefaultObject's on the standalone and pop-up format.](image)

At pop-up format, the visualization of the object depends of the events that occur on the other object. At the standalone format, the object is independent and can be resized, moved, and closed. The Flash application also contains some components, specifically developed to the DIR, that give a more intuitive interpretation of the information generated by DOLPHIN. There are five specific components: Node, CNode, Expression, DTlist and DT. The first two are used to represent the vertices of the CFG (designated simply by nodes); the Expression represents the expressions trees; the DT represents DataTransfer’s (element of the DIR that characterizes the data exchanges); and DTlist, a list of DT. Figure 9 shows an example where appear several of these components. Except for the DT objects, all the others can be moved, resized and closed. Most of the developed components have sensible areas that react to some events produced by the mouse, such as the roll over, roll out and click.
Currently, are under development some new components to visualize temporary information, generated by the analysis routines, that was not contemplated on this solution (it is possible to observe this information with the HTML solution). But the plans for the WMAS are much more ambitious than these ones; the main goal is to build a visual simulator for the DOLPHIN framework, eventually, based on the DVM (DOLPHIN Virtual Machine). But since this one is not yet implemented (it is only on the paper), the next step is to build new Flash components, to simulate the behavior of some analysis and optimizations routines. The drag-and-drop of these components over the objects that are already implemented will graphically reproduce the effects of the routines.

### 4.3 Web Compiler Components Development System

The Web Compiler Components Development System (WCCDS), is the third component of DOLPHIN-FEW and is presently only on the paper. The goal is to build a complete IDE, available via Web, to develop components for the DOLPHIN framework.

This IDE should supply context sensitive editors, to make the interface with DOLPHIN-CCDS tools, namely with the CDD, DBEG, DAOG and DCMS. But also should improve the integration between these tools. The WCCDS should allow to:

- Help the edition of the several specifications, eventually, combine them into a single one;
- Supply templates of some of these specifications, for example, the computer architecture specification of known processors;
- Represent and work with routines, like visual objects;
- Write the DCMS specification using graphical objects;
- Develop new extensions to the DAOG, such as parametric routines (represented graphically);
- Automatically integrate the components on the DOLPHIN framework and test them;
- Simulate the behavior of the components;
• Put the components immediately available to be used.

Most of these objectives can be obtained with the integration of the WIDE and the WMAS with the WCCDS, allowing with an unique application, the development, test and simulation of new components for the DOLPHIN framework.

5 Conclusion

The goal of this paper was to introduce the architecture of DOLPHIN framework, focusing on the DOLPHIN-FEW. The idea around this architecture is quite novel (we do not know nothing similar). We dare to say that is not one more solution. DOLPHIN-FEW takes advantage of some Web technologies, to give a new perspective and, new possibilities for the compilers development. By the other side proposes a new set of Web services.

All the components of DOLPHIN-FEW, have space to grow, particularly, the WMAS and the WCCDS, that we believe will have excellent results and certainly will produce new ideas.

Probably, the biggest obstacle of DOLPHIN-FEW, is the self DOLPHIN framework, that still have many things under implementation and some that are only on the paper. But we believe that now, that the results started to appear, the development time will decrease.

DOLPHIN framework is an ambitious project, but here we just discuss the work directly related with compilers development. There are also some experiences related with the representation and optimizations of problems from other areas, like production management and planning; experiences with declarative languages for application on robot programming; and even some experiences related with the development of tools for pedagogical purpose.

Along the paper many of the future goals of this project were introduced. We still work on the DOLPHIN project to achieve these goals, hopefully to present soon more results.

References