Component-Oriented Software Reengineering using Transformations

Valdirene Fontanette
valdirene@dc.ufscar.br

Vinicius Cardoso Garcia
vinicius@dc.ufscar.br

Ângela de Britto Perez
angela@dc.ufscar.br

Antonio Francisco do Prado
prado@dc.ufscar.br

Computing Department
Federal University of São Carlos
Rod. Washington Luis, Km 235
P.O. Box 676 - 13565-905
São Carlos - SP - Brazil
Fax: +55 16 260-8233

Component-Based Software Engineering
Object-Oriented Software Engineering and CASE
Software Engineering Tools and Techniques
Component-Oriented Software Reengineering using Transformations

Abstract

This article presents a strategy of Component-Oriented Software Reengineering using Transformations, to rebuild legacy systems. Components are used to facilitate the reuse and maintenance of the rebuilt system. The strategy is a result of several researches [1,3,4,5,7].

The strategy is divided in 4 steps: Organize Legacy Code, Recover Project, Reproject and Reimplement. In the Organize Legacy Code step, the legacy code is organized, according to the Object-Oriented principles, partitioning in supposed classes, attributes and methods to facilitate the Component-Oriented Reengineering. In the Recover Project step, it is obtained the legacy system descriptions in MDL (Modeling Domain Language) that will be persisted using UML techniques, in the MVCase tool. The project is recovered in the MVCase tool, starting off with MDL descriptions, and the organized code parts, corresponding to the procedures and functions, candidates to methods of a class, are transformed to Java. In the Reproject step, it is done the Component-Oriented reproject of the project recovered in the MVCase tool, to satisfy the new requirements, following the Catalysis method and the Enterprise Java Beans technique (EJB). Finally, in the Reimplement step, the components description in MDL/Catalysis are transformed into Java beans, and integrated to the Java code of the methods, obtaining the system final implementation.

Key words: Software Reengineering, Reverse Engineering, Object Orientation, Software Transformation, Components and Reuse.

1. Introduction

Nowadays, there are a great number of companies working with systems implemented in old programming languages, whose maintenance is arduous and onerous. Software system is an evolutionary piece and requires constant modifications, or to correct mistakes, to improve performance, to add improvements or even to adapt it for new hardware and software platforms.

Along the years, these systems incorporate substantial knowledge of its context, including new requirements and business rules. Several are the factors that induce a system to fall in disuse. Time is one of these factors, because as it passes, new requirements appear. In most of the cases, these systems, denominated legacy, are of great utility for their users and a lot of times, their reconstruction, using modern software development techniques, could be the solution to their reusing without needing to build a new system.

The Software Reengineering researches modern techniques to accomplish the maintenance in a system in a more automatic and economical way. Jacobson and Lindström [6] define reengineering as the process of creating an abstract system description, to think on a change in a high abstraction level and finally reimplement this system.

Usually, the legacy systems don’t have an updated documentation, or even nor they have documentation, and the person that developed the system is not always the responsible for its maintenance. Factors like these ones turn the
maintenance difficult and of high cost, inducing the developers to leave the system. In these cases, the system reengineering is based only in its source code.

The process of obtaining the system documentation, creating more abstract descriptions, is obtained through reverse engineering.

The Software Reengineering is also a form to obtain reuse and the domain application understanding, recovering the information in the analysis and project stages, organizing them in a coherent and reusing way.

Motivated by these ideas, it was researched a combination of techniques and tools that make possible the recovery of legacy systems, rebuilding them for being executed in new hardware and software platforms. The integrated environment supports the system maintenance, to reproject it component-oriented, guaranteeing its continuous evolution and reusing through components reuse.

2. Main Used Techniques

One of the main techniques of the proposed Reengineering is the software reconstruction using transformations. Different transformation systems have been used standing out the Tampr [14], Refine [15], Popart [16]. Another important transformation system that has been used in this area is Draco, according to [1,3,4,5,7].

Usually, a transformation system restructures a program A in a program B, applying a set of well-defined transformations preserving the semantics of A in B. The Draco machine was built to test, develop and to put in practice the Draco paradigm. It is a transformation system based on the idea of software reconstruction by domain-oriented transformations. According to Prado [7], it is possible to do the software reconstruction by the direct “load” of a language source code to languages of other domains. A domain, according to the Draco paradigm, is composed of three parts: the domain language: used to describe the domain specifications. It has a parser that is generated automatically by the Draco component called pargen starting off with language grammar definitions. The parser has as task to generate automatically the internal program representation, written in the domain language that is manipulated by Draco. This description internal representation in a domain, used by Draco, is denominated Draco Syntax Abstract Tree (DAST); the prettyprinter or unparsr: that accomplishes the DAST formatting, turning it textual again in the domain language. Based on grammar definitions the Draco’s subsystem called ppgen generates, automatically, the respective domain prettyprinters; and one or more transformers that map syntactic structures of a language to other syntactic structures, that can be in the same domain language, called Intradomain transformers, or in another domain language, called Interdomain transformers. The transformers are responsible for the software reconstruction process automation.

Figure 1 shows the transformations execution sequence. Initially, starts off with source code A that, is analyzed by the parser, which is responsible in generating the DAST1.
Applying the Intra or Interdomain transformations, a new DAST2 is obtained. Using the prettyprinter a new source code B is obtained.

Different control points are used in the transformations. Usually, we have the LHS (Left Hand Side) and the RHS (Right Hand Side) that define the recognition and substitution patterns, respectively.

In order to assist the transformations, the Draco transformation system has a Knowledge Base, which allows storing facts and rules with legacy code information, which will be consulted later. The Knowledge Base stores facts related to the supposed classes, attributes, methods and relationships. Another Draco resource is the Workspace, used to compose command blocks starting off with commands dispersed throughout the code. It facilitates the execution of transformations that need information captured by other transformations.

Another technique that stands out in reengineering studies is the one of reverse engineering called Fusion/RE [8,11], used to obtain the understanding and to revitalize the legacy source code structure according to the principles of the object-oriented paradigm, aiming to reuse all the legacy code functionality in the system reconstruction. The strategy presented in this article uses only part of the Fusion/RE approach to recover the Current System Analysis Model (CSAM) with its supposed classes, supposed attributes, supposed procedures and relationships that originally are not object-oriented, and since then, elaborate the objects model of the System Analysis Model (SAM), correcting the procedure anomalies. Besides Fusion/RE, other reverse engineering techniques are used to obtain the Use Cases of the legacy code and of its Sequence Diagrams, which represent the execution flows of the normal and alternative courses of each use scenery of the legacy code, obtained in several transformation experiences, using Draco, such as Cobol to C++ [17], Clipper to Java [1], Progress to Java [5] and Procedural DataFlex to Visual Object-Oriented DataFlex [3].

The different techniques used in the reverse engineering of the legacy code allow recovering models of the first level of Catalysis. Catalysis[13] is a Component-based development method that covers all the phases of the system life cycle, from the specification to the implementation. Catalysis supports from the problem domain analysis and specification to the specification and the components inner project that constitutes the elements reused in different
applications. The software development process in *Catalysis* is divided in three logic levels: **Problem Domain, Component Specification and Component Inner Project**.

The proposed strategy uses the Enterprise Java Beans technique EJB [9,12], to implement the components. The *EJB* technique allows separating the interface, business rules and the database, creating multi-tier applications. This technique uses transaction-oriented components that are executed in EJB servers. According to the *EJB* technique, the components that implement and encapsulate the business rules are available in the *EJB* server.

Besides these techniques, it is standing out the use of CASE tools in the project and reproject of systems to be reconstructed. A CASE tool that has important characteristics to this strategy is the MVCase, result of another research [2] that besides supporting the system specification in the Object-Oriented language UML, generates code automatically in an object-oriented programming language, starting off with high-level specifications, using distributed components.

Combining the ideas of the Draco transformation system, the reverse engineering of Fusion/RE method and the experience in reengineering researches [1, 3, 4, 5, 7], the component development method *Catalysis*, EJB, the MVCase tool, it was defined a strategy for Component-Oriented Software Reengineering using Transformations, presented next.

### 3. Component-Oriented Software Reengineering using Transformations

The strategy of Component-Oriented Software Reengineering using Transformations is accomplished in 4 steps: Organize Legacy Code, Recover Project, Reproject and Reimplement, according to Figure 2.

---

**Figure 2 – Strategy of Component-Oriented Software Reengineering using Transformations**

#### 3.1. Organize Legacy Code

The legacy code, written in a procedural way, has commands and declarations that can be organized, without losing its logic and semantics, in order to facilitate its transformation to the object-oriented paradigm, which has the class as a basic unit. In the first step, **Organize Legacy Code**, the Software Engineer with the Draco transformation system support, organizes the legacy code, obtaining a code still in the same language, however organized according to the Object-Oriented principles.
The Software Engineer joins all the available documentation about the legacy code architecture. He elaborates the CALL/CALLED list [8], which determines the sequence of programs to be submitted to the transformations that organize the legacy code. This manual activity was followed to improve the knowledge about the system, facilitate the transformation definition process, identify the sequence to be followed and to facilitate the transformers implementation.

Firstly, the legacy code is submitted to a transformer, which contains transformations without the substitution pattern (RHS), since its function is to store facts and rules with code information in the knowledge base, which will assist in the solution of problems of legacy code organization. Among these transformations, there are the ones to:

a) Identify supposed classes through the recognition of data file opening and accessing commands. Besides, every command block, which does not refer a data file, is related to the system interface and later will also give origin to a supposed class;

b) Identify supposed attributes, which correspond to data file fields;

c) Identify supposed methods, of a supposed class, starting off from program units that can be procedures, functions or command blocks, maintaining the scope, data dependence the legacy code visibility. A program unit that refers to only a data file, it is related to this data file. If the program unit does not consult nor modify any data file, it is related to the system interface. The supposed methods are classified in constructor (c), when they change the data structure, and observer (o), when they only consult the data structures. When a supposed method refers to more than a supposed class it should be allocated using the following criteria:

- If a supposed method is constructor of a supposed class and observer of another (oc), it will be allocated in the supposed class that refers as constructor;

- If a supposed method is observer of a supposed class and constructor of several (oc+), it will be allocated in the supposed first class that refers as constructor; and

- If a supposed method is observer of more than a supposed class and constructor of just one (o+c), it will be allocated in the first supposed class that refers as observer;

All program units candidate to supposed methods of a supposed class are gathered in a single file;

d) Identify relationships, through variables and commands that manipulate data file data. It is verified if a file field, identified as key, has its values consulted and attributed to a variable, and later that variable value is stored in fields of another file or it is used to search in registers of other files.

The Intradomain transformer, which organizes the legacy source code, uses syntactic patterns of the same language in the recognition (LHS) and in the substitution (RHS) control points. This transformer uses the Knowledge Base used by the first transformer, to assist the transformations to segment the legacy code in atomic procedures,
maintaining the scope, data dependence and the legacy code visibility; to correct procedure anomalies using the Fusion/RE method; allocate the supposed methods, corresponding to the segmented procedures, in a supposed class.

Figure 3 shows a transformation applied by Draco to identify the supposed classes and their supposed attributes.

```plaintext
TRANSFORM UseStat
 LHS: {{dast clipper.statements
     USE [[iden FileName]] INDEX [[iden* Indexes]]}}  (1)
}
POST-MATCH: {{dast txt.decls
    KBAssertIfNew("SupposedClass([[AreaNum]], [[CurrentClass]], [[ClassNum]])"); (3)
    KBWrite("COSRT.kb");

    ... found = ObtainSupposedAttribute(ContAttribute);
    KBWrite("COSRT.kb");
    KBWrite("COSRT.kb");
}}

Figure 3 – Transformation example to identify the supposed classes and their attributes

The UseStat transformation, in the LHS (Left Hand Side) control point (1), defines the recognition pattern for the USE command, which recognizes a data file, in this case, a file with a .dbf extension. In its POST-MATCH control point (2), a “SupposedClass” fact, containing this data file name, is stored in the Knowledge Base through the KBAssertIfNew() and KBWrite() commands (3). The ObtainSupposedAttribute function (4) is used to identify the fields of this data file and store them in the Knowledge Base, through the “SupposedAttribute” fact (5).

Figure 4 shows an Intradomain transformation example, used in the legacy code organization. In the left side stands out the recognition pattern for the reserved word FUNCTION, which recognizes a function in the legacy code. Thus, the legacy code is navigated and every time the reserved word is found, it executes its POST-MATCH control point. The Knowledge Base is navigated in order to find a “SupposedMethod” fact containing this function name and a “SupposedClass” fact corresponding to this supposed method. If it is found, it is created a WORKSPACE containing the class name and the TEMPLATE, containing the substitution pattern RHS (Right Hand Side) for the reserved word FUNCTION is called. Then, the TEMPLATE content is allocated in the WORKSPACE.

```plaintext
TRANSFORM FuncDecl
 LHS: {{dast clipper.prog_extra
     FUNCTION [[iden FuncName]]
         [body_struct* Commands]
     RETURN [[expr Return]]}}

POST-MATCH: {{dast txt.decls
    TEMPLATE "TFunc"
}}

Figure 4 – Transformation example to organize the legacy source code

Concluded this step, we have the organized legacy code following the object-orientation principles.

3.2. Recover Project

In the second step, Recover Project, the Software Engineer starts off with the organized source code and again with the Draco transformation system support, obtains the system MDL description. Using the MVCase tool, the Software Engineer imports the MDL description to obtain the current system recovered project. It is obtained the class diagram with the system classes, their respective attributes and methods, and the relationships among the classes.
Figure 5 shows an Interdomain transformation example, which transforms a supposed class into a class and generate a MDL description for a class and its attributes. The LHS (1) shows the recognition pattern to identify a program through the Progname metavariable. In its POST-MATCH control point (2), the knowledge Base is navigated and the first “SupposedClass” fact (3) is consulted and the supposed class name is recovered. Then, a “Class” fact (4) is stored in the Knowledge Base.

<table>
<thead>
<tr>
<th>TRANSFORM Program</th>
</tr>
</thead>
</table>
| LHS: {{dast clipper.program  
|   [[TAG tag]] [[ID ProgName]] (1)  
|   [[prog_elements Elements]]}}  
| POST-MATCH: {{dast txt.decls  
|   while (KBSolve(query)) { (3)  
|     KBAssert("Class([[ClassName]], [[ClassNum]])"); (4)  
|     KBWrite("COSRT.kb");  
|   }  
|   if (KBSolve(query)) { (5)  
|     ...  
|   while (KBSolve(query)) { (6)  
|     TEMPLATE("Class") (8)  
|       TRANSPORT_VALUE("NameWithAspas");  
|       TRANSPORT_VALUE("NumWithAspas");  
|       MOVE1(ClassOper, "COper");  
|       MOVE1(ClassAttr, "CAttr");  
|       PLACE_AT(WkspClass[ClassNumber]);  
|     END_TEMPLATE;  
|   }  
| | |

Figure 5 – Transformation example to generate the MDL description of a class and its attributes

The Knowledge Base (5) is navigated one more time in order to find a “Class” fact containing the name of the program to be recognized. A “SupposedAttribute” fact (6) for this class is looked for. This step is done until all the attributes of the class are found. Information about the supposed attributes such as: name, type, size are recovered and temporarily stored in a WORKSPACE through the TEMPLATE Tattribute (7). The attributes and the class operation descriptions, stored in WORKSPACES, are moved to another WORKSPACE, which contains its class descriptions, through the TEMPLATE TClass (8).

Figure 6 shows the generation of the MDL description of a supposed method. Firstly, the Knowledge Base is consulted to know if the supposed method is triggered by the menu (1) or related to the system interface (2). The

<table>
<thead>
<tr>
<th>TRANSFORM Program</th>
</tr>
</thead>
</table>
| LHS: {{dast clipper.program  
|   [[TAG tag]] [[ID ProcPrinc]]  
|   [[prog_elements Elements]]} [TAGFIM Tag]}  
| POST-MATCH: {{dast txt.decls  
|   scanf(query, "SupposedMethod(%s, %s, *x)", "MenuInterface", ClassName);  
|   if (KBSolve(query)) { (1)  
|     scanf(query, "SupposedMethod(%s, *y, %d)", "SystemInterface", Cont);  
|   if (KBSolve(query)) { (2)  
|     TEMPLATE("TOperation") (3)  
|       TRANSPORT_VALUE("NFnction");  
|       TRANSPORT_VALUE("NId");  
|     END TEMPLATE;  
|   }  
| | |

Figure 6 – Transformation example to generate the MDL description of a supposed method
TEMPLATE TOperation contains the MDL description in its substitution pattern.

After generating the MDL description, the Software Engineer imports it in the MVCase tool to obtain the current system recovered project. Besides the class diagram, it is obtained the sequence diagrams, that show the interactions and the message connections among the system objects, representing their execution flow.

3.3. Reproject

In the third step, Reproject, the Software Engineer, using the MVCase tool, does the Component-Oriented reproject of the recovered current system., The initial modeling obtained in the previous step, is used as base for the specification and project of components. The component specification is based on the Java/EJB technology, using Catalysis as a development method.

For a better idea of this step, it is presented below the details of Catalysis, to create the Customer component

3.3.1. Problem Domain

This level, says “what” the system should do to solve the problem. It is identified the object and action types, gathering them in different visions by business areas.

The Software Engineer models use case diagrams, indicating the actors' relationship with the system, leaving the details that are not relevant, for the context. Figure 7 shows “Customer” actor interacting with the system through the “AddCustomer” use case.

![Use Case Diagram](image)

**Figure 7 - Use Case Diagram**

3.3.2. Component Specification

In this level, the specifications, from the previous level, are refined, emphasizing the components identification, behavior and responsibilities. New models, more detailed, are obtained, but still without worrying about the implementation.

Starting off from the Use Case Diagrams, are built the Sequence Diagrams, which have as objective to show the operation execution sceneries along the time. In the case of the strategy, the Sequence Diagrams, obtained in the Recover Project step, are refined, and new diagrams can be built to satisfy the new requirements added to the project.

Figure 8 shows a Sequence Diagram showing Customer interactions with the system objects.
3.3.3. Component Inner Project

In this level, it is made the component inner project, giving emphasis to their implementations and physical distributions. In the case of the strategy, the EJB technology is used to construct components starting off from the classes obtained in the Recover Project step. To refine the models, the Software Engineer can change the name of the classes, their attributes and methods turning them more significant.

Figure 9 shows the CustomersEJB component inner project. Starting off from the Sequence Diagram and the Use Case Diagram, are created the Remote Interface, that has methods responsible for business rules, and the Home Interface, that has methods to refer and to maintain the remote object. In the case of the Customers bean, it has the CustomersEJB class that implements the EJB Interface and makes available the Customers Remote Interface, that extends the EJBOBJECT Interface, and the CustomersHome Interface that extends the EJBHome Interface. The interfaces EJBOBJECT, EJBHome, and EntityBean are from the javax.ejb [12] package. The “EJB” suffix is added to the name of the class that creates the bean.

3.4. Reimplement

Finally, in the fourth step of the strategy, Reimplement, is made the system final reimplementation in an Object-Oriented language, Java in this case. The Software Engineer uses the MVCASE tool as mechanism of code generation, starting off from class diagrams with their attributes and prototypes of specified methods.
Figure 10, shows in the MVCase tool, how the Software Engineer, makes the Java code generation, starting off from the component specifications.

![Figure 10 – Java code generation based on the component specification](image)

A first experience [1] accomplished following the proposed strategy started off with legacy systems in Clipper [10], whose reengineering had Java as the reimplementation target language and UML as modeling language for system recovery and reproject.

The presentation of a case study that shows the use of the proposed strategy is proceeded.

4. Case Study

It is about a system for Auto-electric and Mechanical Garage that controls the services executed in the vehicles and the stock of the used pieces. The customer goes to the garage to request a service in his vehicle. A customer can have several vehicles. The same vehicle can return to the garage several times, elaborating, in each one of them, a distinct Service Order. This Service Order contains data about the customers, vehicles and the repairs to be done.

When the vehicle is repaired, the Service Order is completed, filling in the used pieces and the executed labor. A lot of times, the repair can demand pieces no existent in the stock, which are acquired out of the garage and also appears in the Service Order. The registration of those pieces is important for the garage manager, because those are candidates to be stocked in the future. The possible vehicle models must be registered by the system, because tables are used for service charge, such as electric as mechanic, according to the vehicle model.

This system [18] was developed in 1990 and has about 20,000 code lines in Clipper.

A presentation of each strategy step is followed for this system reengineering.

4.1 – Organize Legacy Code

Firstly a CALL/CALLED list of programs was elaborated, to determine a sequence of programs to be submitted to the Clipper domain transformers in the Draco transformation system.
Soon afterwards these files were submitted to the Intradomain transformer ClipperToKB, in the Draco transformation system. The ClipperToKB transformer implements transformations to collect legacy code information such as supposed classes, attributes, relationships, currents system interface and its control flow and store them as facts in the Knowledge Base. During the application of those transformations, it can be necessary the interaction with the Software Engineer for the identification of the relationship type and their cardinalities.

In the example of Figure 11 it was identified the procedures (CLEAR, WARNING and WINDOW) that don't refer to any data file. When the transformer identifies procedures like those, it relates them to interface procedures in the Knowledge Base, through the "Procedure" fact and its "SystemInterface" parameter.

It was also identified the data files SERORD and CUSTOMER, that give origin to the corresponding "SupposedClass" facts. The first parameter of the "SupposedClass" fact indicates the number of the work area that Clipper programs use for each opened data file, determined by SELECT command. The third parameter is a sequential number to aid searches in the Knowledge Base. When the USES command is followed by the INDEX clause, the "IndexArea" fact must be stored in the Knowledge Base to represent the index file related to the identified data file.

After this, the ClipperKBToClipper transformations were applied. The ClipperKBToClipper transformer automates the obtaining of the Object Model, with its attributes and methods. Program units that contain anomalies should give origin to new supposed methods. The new supposed methods generate facts containing information about these supposed methods in the knowledge base. The Organize Legacy Code step has a file, with a .PRG extension, as an output, for each supposed class. The procedures and functions that don't refer to any data file are moved to the file "Interface.prg", as shows Figure 12, that gives origin to a supposed interface class.

4.2. Recover Project

In this step the Software Engineer also uses the Draco transformation system to generate automatically the MDL descriptions and the Java code of the methods in each class.
The MVCase tool is also used to import the MDL descriptions and to recover the current system object-oriented project specifications in UML and Java.

To obtain the system object-oriented model, all programs organized in the previous step, are submitted to the ClipperToUMLJava transformer.

Figure 13 shows the generation sequence of a supposed class, from the legacy code information stored in the Knowledge Base for the legacy code organization (1), the MDL description (2) that imported in the MVCase tool recovers the system project as the class diagram (3.1), which contains Java description built-in the semantics of the class methods (3.2).

Figure 12 – Organized legacy code: Interface.prg

Figure 13 – Class MDL Description and its Visual Diagram imported in the MVCase tool
4.3. Reproject

With the project recovered in UML, it becomes easier to understand and reproject it to satisfy the new requirements. It is also possible to edit and to change the methods semantics, "loaded" directly to Java. The Software Engineer uses the Catalysis method to specify new requirements, refine the models recovered in the previous step, project the components, and apply the EJB technology to restructure the system in multitiers, making available components that can be reused in other systems.

Figure 14 shows the AddServiceOrder (1) Use Case and its details specified in the Sequence Diagram (2), models corresponding to the Problem Domain and Component Specification levels of the Catalysis method, respectively. The Class Diagram (3), of the Component Inner Project level, shows the components specified with the EJB technique.

4.4. Reimplement

In the last step of the strategy, the MVCase tool automatically implements the project recovered and specified in UML to the Object-Oriented language Java. The class diagrams of the reprojected system are reimplemented in Java. Figure 15 shows a new system interface reimplemented as a Java Frame.

The generated code can be executed and the execution results are analyzed by the Software Engineer to verify if the rebuilt system satisfies the old and new requirements.

Thus, it is obtained a feedback of all the strategy steps, guiding in the possible corrections and validation of the Object-Oriented system.

The legacy system functionality is maintained in the reimplemented system, with the great advantage of facilitating the maintenance, once the code is organized and encapsulated in classes and objects. Besides, the use of the EJB technology incorporates to the system a great flexibility and an organized structure in components, which can be reused directly in new projects or through some adaptations.
5. Conclusion

This article presented a strategy to transform Procedure-Oriented source code of legacy system, to Component-Oriented systems, to be executed in modern platforms of hardware and software. When emphasizing the project and the construction of systems through the components reuse, it is obtained considerable benefits in relation to the software quality, the software engineer’s productivity, the system development cost reduction and easiness in the maintenance.

To support the legacy system reengineering, the strategy integrated different techniques, standing out the Reverse Engineering, a CASE tool for the import and reproject of the system modeling obtained by the transformations, the Catalysis method, the Enterprise Java Beans (EJB) technology and the Draco transformation system. The strategy can be applied to legacy systems implemented in different languages as, for instance, Cobol, Procedural Dataflex and Progress 4GL, because the Draco transformation system offers the capacity to work with a diversity of domains. With the obtained results, it was proven the viability of transforming legacy systems in Component-Oriented systems through the integration of different techniques and approaches with the transformation system. Besides the systems updating, the strategy also makes possible the project recovery and its modification to satisfy the new requirements.

References