Orion-RE: A Component-Based Software Reengineering Environment

Alexandre Alvaro, Daniel Lucrédio, Vinicius Cardoso Garcia, Antonio Francisco do Prado, Luis Carlos Trevelin

Federal University of São Carlos / Computing Department
{aalvaro,lucredio,vinicius,prado,trevelin}@dc.ufscar.br

Eduardo Santana de Almeida

Federal University of Pernambuco / Informatics Center / Recife Center for Advanced Systems Studies
esa2@cin.ufpe.br

Abstract

Software reuse is the process of implementing or updating software systems using existing software assets, resulting in a software quality increase, productivity and reduced time to market. One way to achieve reuse is through software reengineering. This papers presents Orion-RE, a Component-Based Software Reengineering Environment that uses software reengineering and component-based development techniques to rebuild legacy systems, reusing their available documentation and knowledge embedded in the code. Orion-RE integrates several tools: a software transformation system, modeling/development tools, a software artifacts repository, and a middleware platform. A software process model drives the environment usage through the reverse engineering, to recover the system design, and forward engineering, where the system is rebuilt using modern technologies, such as design patterns, frameworks, CBD principles and middleware.

1. Introduction

Software reuse is one of the most researched subjects in Software Engineering. The software systems construction using existing software assets, which results in a software quality increase, productivity and reduced time to market, is very attractive to software producers. There are two areas which seek reuse: Software Reengineering and Component-Based Software Engineering.

Software Reengineering, also known as renewal or recovery, has as the main objective the reconstruction of legacy systems to increase their quality and maintainability [38]. One important aspect of Software Reengineering is that it doesn’t only recover the information of an existent project, but it also uses it to alter or rebuild the system, adding new requirements or introducing new technologies. The main activities in Software Reengineering are: reverse engineering, followed by changes in the system (functionality or implementation changes) and followed by forward engineering. In other words, “reengineering is the process of creating an abstract description of the system, elaborating changes in high abstraction level and then implementing the system” [20].

Aiming to achieve larger software reutilization during software systems reconstructing, the use of Component-Based Software Engineering (CBSE) techniques in reengineering is being researched [12, 30]. CBSE is a process that emphasizes the design and construction of computer-based systems using reusable software “components” [38]. In other words, it aims the construction of reusable software parts, which can then be composed to form an application.

The combination of Reengineering and Component-Based Software Engineering techniques can be a solution to rebuild existing systems, reusing the knowledge that has been accumulated during their usage over the years and delivering more evolvable and maintainable systems.

The use of automated tools in Software Reengineering speeds up the process and helps to assure quality in the process. However, there is a lack of good tool support for the CBSE activities, mainly because most of the tools does not offer comprehensive, integrated support for the full range of CBSE activities [27]. The same happens in Software Reengineering, where there is a lack of tools that perform reengineering using CBSE techniques.

Motivated by these ideas, this paper presents a CASE tool environment, called Orion-RE, which combines Software Reengineering and Component-Based Software Engineering techniques, giving support to Component-Based Software Reengineering (CBSR). The construction of this kind of environments, which are referred in this paper as Component Based Software Reengineering Environments (CBSRE), involves issues relating to CASE environments in general, such as tools integration, and issues related to CBSR, such as reverse engineering and component-based development techniques.

Orion-RE is an extension of a Component-Based Software Engineering Environment, called Orion [25], and
integrates different works from our research group, including a transformational system, an UML-based modeling tool, a Java-based programming tool, a network tool, a repository and a middleware platform. A process model, called SRT (Software Reengineering using Transformations) [13], guides the Software Engineer through the usage of Orion-RE, consisting of two major activities: reverse engineering and forward engineering. The first activity aims to recover, through software transformations, the system’s design, with basis on the source code and the available information. The second activity aims to rebuild this system using distributed software components. The use of reusable components, together with distribution principles, helps to make the system more flexible, reliable and with better performance, among other benefits [11].

Although these works were not originally designed together, they could be integrated into an environment that fulfills most of the requirements for a CBSRE.

The paper is organized as follows: Section 2 presents the main requirements to CBSREs. Section 3 presents the different works from our research group that compose the Orion-RE environment. Section 4 presents the Orion-RE environment. Section 5 revises the Orion-RE environment with the identified requirements. Related works are presented in section 6. Section 7 presents some conclusion remarks.

2. Requirements for Component-Based Software Reengineering Environments

It is a difficult task to enumerate a definitive set of requirements for CBSREs, because the complexity of the involved factors. However, several researches found in literature [9, 10, 38] attempt to identify requirements related to Component-Based Development and Reengineering Environments. Thus, basing ourselves in literature and in our experience [13, 16, 30, 33, 34], we identified six requirements for CBSRE:

- **Language and technology independence**: A CBSRE can’t be fixed in a limited set of languages and/or technologies, since its applicability would be compromised. Legacy systems are written in many different languages, and there are countless new technologies that a reengineering work may focus in.

- **Extensibility**: Since new technologies are constantly arising, a CBSRE must be able to incorporate these technologies to be used in reengineering. This requires an extension mechanism, responsible for keeping it updated in relation to recent technologies.

- **Scalability**: Legacy systems tend to be huge [9, 10]. Therefore, a CBSRE must have support to processing a large number of elements during reengineering.

- **Support for Reengineering activities**: A reengineering process involves several activities, including:

  - **Information gathering**: All the existing information and documentation must be reviewed and, if needed, updated. This helps to assure a clear understanding of the legacy system;
  - **Reverse engineering**: This activity aims to obtain, starting from the source code, a high abstraction level representation of the system, recovering process and data information. It is very useful in the cases where there is no available documentation;
  - **Forward engineering**: This activity is responsible for rebuilding the system, to add new functionalities and/or to improve quality and performance.

- **Tool Integration**: As any CASE Environment, a CBSRE is composed by a set of tools that must interoperate in order to help the Software Engineer through the reengineering process. This interoperation can be achieved in different levels, Platform, Data, Presentation, Control and Process integration [43];

- **Reusability**: Since a CBSRE aims to rebuild systems in a component-based approach, it provides means to assure that the produced components can be later reused;

- **Components Publishing**: In order to effectively achieve reuse, there must be a way to make the produced components publicly available to be consulted, retrieved and reused; and

- **Compositionality**: After recovering the components, the Software Engineer must be able to compose them into applications.

3. Components of the Orion-RE Environment

Our research group has proposed several works in the software engineering area [2, 6, 17, 23, 32, 44]. These works were combined into a CBSRE, meeting most of the identified requirements. Next, these works are briefly discussed.

3.1. MVCASE

The MVCASE [2] (Multiple View CASE) is a Java-based, platform-independent modeling tool, providing graphical and textual techniques based on UML notation [41].

By supporting UML, MVCASE allows the Software Engineer to specify systems in different abstraction levels and in four different views: Use Case View, Logical View, Component View and Deployment View.

To persist the UML specifications, MVCASE uses the same format used by the commercial tool Rational Rose [40] and/or the OMG’s XMI (XML Metadata Interchange) standard [35]. The XMI is a XML-based format used to represent UML models. Based on the four views cited above, MVCASE generates XMI descriptions that can be physically stored, in a file or a repository.
In order to support the implementation activities, MVCASE provides wizards to automate tasks such as code generation, packaging and component publishing. Until this moment, these wizards support the following technologies: Java, EJB [46], JAMP [18], CORBA [36] and Java Servlets [47]. MVCASE is currently being used in several Brazilian institutions, serving as support to other researches and for teaching. More details about the tool can be found in [2; 25].

3.2. JADE

JADE (JAVA DESIGNER), is a Java-based, object-oriented programming tool with features to help the Software Engineer in the coding activities. Similar to most of the commercial programming tools, such as JBuilder [8] and Delphi [7], JADE has support for GUI (Graphical User Interface) design, coding, compiling, executing and debugging applications.

The major different between JADE and most of the commercial tools is its ability to persist the edited classes in XMI format. JADE has a Java parser which retrieves the information directly from the code, identifying the classes information, such as inheritance, attributes names and operation signatures. Based on these elements, XMI descriptions are generated. The GUI information is also persisted in this format, using an extension of XMI that was constructed with this objective.

The use of XMI allows the view of recovered information from the code by modeling tools that can read XMI, such as MVCASE, for example. JADE is currently a prototype under development, being part of an ongoing MSc. dissertation [32].

3.3. MoDPAI

MoDPAI [6] is a tool to monitor devices using pervasive computing issues and intelligent agents. Pervasive computing [21] is the technology that allows the access to Internet through mobile devices, such as digital cellular phones, PDAs and pagers, among others. MoDPAI uses software agents [24] to help the network administrator in his decision-making, speeding up the monitoring and devices control. The exchange of information between the administrator and agents can take place directly in the tool or through mobile devices connected to Internet.

MoDPAI meets the specifications described by the SNMP (Simple Network Management Protocol) [29, 45], a network management standard based on TCP/IP [49]. By meeting the SNMP standard, MoDPAI tool can monitor a network to collect information about distributed devices. This information is analyzed by the software agents that are programmed in a knowledge base to manage the monitored devices. According to the degree of intelligence, which is predefined by means of clauses written in the KB language [28], the agents make decisions that change the devices configurations. Alternatively, the network administrator may interact directly with the tool through its interface and menus.

In order to be able to monitor a network, the administrator must identify and register all the devices to be monitored. MoDPAI provides mechanisms to automatically search the network for devices, identifying the network topology. This search is performed either by looking up devices that have the SNMP management software installed, or by looking up devices that respond to PING requests [29, 45]. After identifying the devices, MoDPAI exports the network topology in XML descriptions. Figure 1 shows how it is performed.

![Figure 1. Automatic devices identification.](image)

This ability to identify the network topology, and export it to XML documents makes MoDPAI an important part of Orion-RE, as it will be explained later.

MoDPAI tool was the subject of a MSc. dissertation [6], and has several other functionalities, related to network monitoring, such as agents programming and reports generation, but these are not related to this paper.

3.4. JAMP Platform

The JAMP Platform (Java Architecture for Media Processing) is an environment for the development of cooperative multimedia applications [18, 44]. It is composed by several servers, a broker (a specialized server that serves as a bridge between different clients and servers) and a framework set that is used by multimedia applications. Based on the object-oriented paradigm, JAMP uses the Java language, the RMI [48] and CORBA models for the construction of cooperative multimedia applications, built on the top of it.

The JAMP platform architecture has three layers. In the first layer, called Application Layer, there are several cooperative multimedia applications. These applications use the services available in the platform through the access frameworks. The second layer, called Services Layer, comprises all the multimedia and distributed middleware services, including: Naming, Trading, Security, Notification, Midi, Collaborative, among others. In this layer, the JBroker service is also located, that provides important functionalities, allowing clients to
invoke methods on remote objects without knowing location or implementation details.

The third layer, Infrastructure Layer, provides communication transparency for the distributed applications and the services. Java/RMI and CORBA are examples of the models supported by this layer. The physical network elements are also placed in this layer.

Figure 2 shows JAMP usage. The process starts with the server registering itself in JBroker (1). When a client needs a service, it accesses the JBroker (2) to search for it. Next, the JBroker returns to the client a reference (3) for an object that implements the service. The client can then invoke the service directly on the server (4). This process is known as Trading Process.

JAMP is the result of several MSc. dissertations [19, 44], being used by our research groups and constantly improved with new functionalities.

3.5. Repository

The software development process involves a lot of different kinds of information. These include requirements, graphical analysis and design models, drafts, and executable code. These information constitute an important knowledge base of the software development process, influencing directly in its quality.

In CBSE, the management of this information is even more important, because reuse, one of the CBSE premises, requires efficient storage and search mechanisms.

Motivated by these ideas, a repository was proposed to remotely store, search and recover software artifacts that are produced during the development process.

The repository is based on the XMI standard. Since scope of the information that can be represented in XMI is not fixed, the repository extends it to represent other kinds of information, such as project execution information and version control. Figure 3 shows the architecture of the software artifacts repository.

The repository has mechanisms for version control, which means that it can control different versions of a given artifact. It has also security and transaction management mechanisms, to assure the consistency and reliability of the stored artifacts.

The services for storing and searching software artifacts are available through a middleware. By doing so, these services can be accessed by a network. The artifacts are transferred in XMI, being physically stored in this format. By doing so, any tool can use the repository, needing only to be able to import and export in XMI, and to have access to the middleware layer. Currently, this middleware is the JAMP platform, described in section 3.4.

Using the repository, the information produced during the software development process can be stored and managed, becoming available for later reuse. The repository is currently under development, using multi-agents systems technologies and search engines, being part of an ongoing MSc. dissertation [26].

3.6. Draco-PUC and DDE Tool

Among the different Transformational Systems (TS) [20, 50] used to automate part of the Software Reengineering process, Draco-PUC [23] stands out, being used in several research projects [12, 13, 16, 23, 30, 31], to implement the ideas of domain-oriented software transformation in Software Reengineering.

Draco-PUC is a generic transformational system that contains most of the important characteristics of a transformational system. Among these characteristics, stands out a parser system, based in Bison [14], allied to other resources, such as a Knowledge Base (KB) [28] and backtracking [15]. The transformations in Draco-PUC can be written directly in the syntax of the languages being transformed.

Several researches were developed to prove Draco-PUC’s viability in the software transformation involving different domains. These can be found in [1, 4, 37].

During its usage, it was observed that one of the great difficulties in the construction of the domains in Draco-PUC relates with the lack of appropriate tools to edit and depurate grammars, transformers and scripts to apply the transformations. Given these needs, the Draco Domain Editor (DDE) [17] was built. DDE allows the Software Engineer to edit grammars, transformers and scripts. The scripts can be executed, to apply the transformations. DDE also supports projects administration. Integrated with Draco-PUC, DDE facilitates the construction and usage of transformations, allowing to take full advantage of this powerful transformational system.

3.7. A Reengineering Process Model

Recently, with the support from RHAE/CNPq [3] and DC/UFSCar [42], a Software Reengineering process model was developed to aid the Software Engineer in the
reengineering of legacy systems. The Software Reengineering using Transformations (SRT) process model [12, 13, 30] uses Draco-PUC, the MVCASE tool, and other technologies to rebuild legacy systems using software transformations.

The SRT was divided in two phases: Reverse Engineering and Forward Engineering, as shows Figure 4.

In the first phase, the Software Engineer starts from the legacy code and all the available information to obtain the legacy system design, in high-level abstraction models. Using transformers to apply reverse engineering techniques, the design elements are identified and organized according to object-oriented principles. Next, these elements are represented in a modeling language.

In the second phase, the Software Engineer starts from the object-oriented specifications recovered in the first phase, and can re-specify the system using a component-based approach, adding new requirements and/or modifying the programming language. In this way, a new design of the system, which is component-oriented, is obtained. Next, these components are implemented, producing a new system implementation. Finally the re-implemented system is deployed.

If not modified during the reengineering process, the existing functionalities of the legacy system are maintained, with some benefits such as greater reuse degree and easier maintenance, due to the code organization obtained from the component-based approach.

Case studies were conducted in order to evaluate the SRT method. The results, as shown in [16, 22, 33, 34], indicate that some gains are achieved with SRT, such as the partial automation and documentation recovery.

4. Orion-RE Environment

Section 3 presented several works from our research group. In this section these works are integrated into a CBSRE. Orion-RE covers a substantial part of the software reengineering lifecycle, with tools that aid the Software Engineer in the reverse and forward engineering activities. Figure 5 shows the environment architecture.

Guided by the SRT, the Software Engineers use the tools in different tasks, such as design recovery, re-specification and code generation, for example. The artifacts produced by the tools during the process are stored in the repository, which has mechanisms to guarantee its consistency and integrity. The tools and the repository are integrated through the services of the JAMP platform, thus they operate over a network.

This architecture allows cooperative work, provided that multiple Software Engineers can work together in the same project, using different workstations and sharing information during the process, through the repository.

Each tool of the environment has specific features, and contributes with the Software Engineer in different ways. However, the most important component of the environment is the SRT, which guides the Software Engineer during Orion-RE usage, providing a well-defined approach to accomplish the software reengineering process. For this reason, the steps of SRT will be followed to explain how Orion-RE works.

To facilitate the understanding of inputs, outputs and controls when using Orion-RE, an Auto-Electric and Mechanics System of a computer company is used as example. The legacy code of the system was in Clipper language and it was rebuilt in Java. The Auto-Electric and Mechanics System applications are divided into three big modules. The first one, Service, is responsible for registering and controlling service tasks. The second one, Stock, is responsible for controlling the sale and purchase of products to accomplish the maintenance in the stock. The third one, Service Order, is responsible for controlling the sale and purchase of products to accomplish the maintenance in the stock. The third one, Service Order, is responsible for controlling service order tasks and emitting reports related to accomplished and pending service order tasks.

As already seen, the SRT is divided in two stages. Following is a detailed presentation of each stage and how the environment is used in each one.

4.1. Reverse Engineering

The Reverse Engineering phase is performed in three steps: Identify, Organize and Recover, as described next. The main mechanisms that help the Software Engineer in this phase are the MVCASE Tool and Draco-PUC.
4.1. Identify. The objective of this step is to identify in the legacy code the elements that will compose the object-oriented models of the legacy system. The Software Engineer extracts information about the legacy code structure, identifying elements such as: the supposed classes, their supposed attributes, supposed methods, relationships, the supposed use cases, and the hierarchy calls, storing them in the Knowledge Base (KB). Figure 6 shows an example of this process, where a piece of legacy code is inspected and all the useful information is identified and stored in the KB. This information aids the Software Engineer in the code organization, according to the Object-Oriented principles.

4.1.2. Organize. The objective of this step is to organize the legacy code, according to OO principles. The whole legacy code is organized in classes with their methods and attributes. This step has the support from Draco-PUC. The organization of the legacy code will be performed in an automatic way, through transformations, with basis on the information recovered in the previous step and stored in KB. Figure 7 illustrates this process, where the legacy code is organized, with basis on the original legacy code and the information that is stored in the KB. Once this step is concluded, the Software Engineer obtains the organized code according to the OO principles. This organization facilitates the task of recovering the object-oriented design of the legacy system, which is performed in the next step.

4.1.3. Recover. In this step, the Software Engineer obtains the legacy system design, with support from Draco-PUC. The design is obtained from transformations that analyze the organized code, together with the information from the KB, being later described in MDL [40], which is the same language used by MVCASE and Rational Rose tools.

Finally, the Software Engineer, using the MVCASE tool, imports the MDL descriptions to obtain the recovered design, as shows Figure 8, where the recovered design is represented by a Classes Model [41], with their attributes, methods and relationships.

Once the legacy system design is recovered, the Software Engineer goes to the next phase of the reengineering, where the system is rebuilt using a component-based approach.

4.2. Forward Engineering

The Forward Engineering phase is composed by four steps, Re-specify, Re-design, Re-implement and Deploy application, which are described next.

4.2.1. Re-specify. In this step, the Software Engineer imports the object-oriented specifications from the legacy system, recovered in the Reverse Engineering phase, in the MVCASE tool. Next, he re-specifies the recovered object-oriented specifications, in MVCASE, to include new requirements, modify the existing functionalities or add new ones. The Software Engineer can create and/or modify the classes models of the recovered design, to represent the new specification.

4.2.2. Re-design. In this step, the Software Engineer, with MVCASE, refines the models from the previous step, defining the components and the physical architecture of the system, obtaining a new design. The Software Engineer worries about the non-functional requirements, such as physical components distribution, persistence, implementation language and the execution platform.
Starting from the models recovered in the previous step, the Software Engineer may, for example, use MVCASE to specify component models [41], where the organization and dependencies between the components are shown.

In this step, the Software Engineering may use components from existing frameworks, allowing the reutilization of components that deal with non-functional requirements, such as database persistence or distribution, for example, facilitating the design task [25].

Once the system is re-designed in a component-based approach, and the functional and non-functional requirements are treated, its executable code can be generated.

4.2.3. Re-implement. The re-implementation can be made automatically through the MVCASE tool, which can generate code in an Object-Oriented programming language, with basis on the models from the previous step.

To perform this step with MVCASE, first the Software Engineer uses a code generator from MVCASE to generate part of the components executable code, such as the classes hierarchy, with attributes, operation signatures and relationships. The components design and the generated code are stored in the repository, in XMI format.

Next, to complement the partially-generated code of the components, the Software Engineer retrieves it from the repository with JADE tool. As mentioned earlier, the major advantage of using JADE instead of the similar commercial tools is that JADE can write high-level XMI descriptions, which can then be read by a modeling tool. By editing the partially-generated code with JADE, the Software Engineer can update the models just by modifying the corresponding code, maintaining the consistency between the design and the executable code.

After this step, all the components are implemented in a programming language, being stored in the repository together with their design. The next step is to deploy them.

4.2.4. Deploy System. Once all the components of the reengineered system are implemented with MVCASE and JADE tools, the Software Engineer goes to the next step, where the components are deployed. Since the components will be distributed, this step involves choosing in which network computer each component will be deployed, aiming to reduce network traffic, maximize the performance, achieve fault tolerance and load balancing, among others issues. Orion-RE environment offers a novel approach to perform this task, as shown in Figure 9.

First, the Software Engineer retrieves the network topology, using MoDPAI tool (1). Next, a XML document containing the network topology description is generated (2). This document is then imported in MVCASE (3), becoming available in the form of an UML’s deployment diagram. Next, the Software Engineer can analyze the best distribution possibilities and, in a “drag and drop” process (4), configure into which computer each component will be deployed. Once the configuration is defined, the Software Engineer use the JAMP services (5) to remotely register each component in its corresponding computer (6).

The use of Orion-RE in the deployment process gives the Software Engineer the ability to automatically configure the execution environment. He can change between different deployment configurations, and decide which one results in better performance. Fault tolerance and Load Balancing can also be configured in the environment, by “dragging” the same component into more than one computer of the topology. JAMP’s ability to register multiple copies of a single server can be used to implement these issues.

4.3. Orion-RE Evaluation

In order to evaluate Orion-RE, several case studies were performed [16, 22, 33, 34], standing out an Enterprise Resource Planning (ERP) system. This system contains over 1,500 source programs, totaling about 1,500,000 lines of code in the Progress language [39], with the following modules:

- Basic: basic systems structures, and the common control and security information;
- Financial: bills to pay, bills to receive, cash flow, cash checking account and banking;
- Materials: purchasing and stocking;
- Sells: selling, revenue, mix, quotas and representatives checking account;
- Production: MRP II, PCP, manufacture control;
- Costs: Budget planning, accounting and industrial cost; patrimony control and fiscal books
- Human resources: payroll, hours control and benefits;

4.3.1. Methodology. The ERP system was developed in the Progress programming language and it was rebuilt using Java as implementation language. The case study was performed in a Brazilian company, called MegaSys Informatics Ltda company, involving 3 subjects.
In the first phase of the case study, Reverse Engineering, Orion-RE was used to recover the system’s object-oriented design. The transformers to recover the OO design starting from the source code in the Progress language were constructed in a MSc. dissertation [34].

After the OO design was recovered, the system was re-specified, re-designed, re-implemented and deployed, using JAMP as the distribution platform. The reengineering process was performed by dividing the system in modules, in a “divide-to-conquer” approach, which facilitated the whole process.

The rebuilt system is being currently used by the company, and the information obtained during the process was recovered and analyzed. Before the reengineering, the only available documentation of the system was a depreciated ERD. After the reengineering, new models could be obtained, such as a new ERD, use cases models, sequence and classes diagrams, among others.

4.3.2. Discussion. After this study, some key points could be identified. When comparing to an ad-hoc approach, the following benefits are achieved when using Orion-RE:

- **Reengineering time:** Even without any experiment to prove these facts, it can be noticed that the reverse and forward engineering tasks, when partially automated by Orion-RE, are executed in less time than if done manually. For example, in this case study, 7 programs, containing 1,000 lines of code each one, have its design completed recovered in about 1 minute. These measurements do not include the time to refine the recovered design, but compared to a manual process, the gains are considerable. Other activities, such as re-specification and re-implementation, are also accelerated through the automation provided by Orion-RE.
- **Reuse:** The use of a distributed repository, with mechanisms to store, search and recover software artifacts, facilitates the reuse of the artifacts produced during the development process; and
- **Maintainability:** The Component-Based approach used by Orion-RE facilitates changes to be performed. The support for code generation, components packaging and publishing helps in evaluating the impact of these changes in an execution environment.

Also, some disadvantages could be observed:

- **Transformers construction:** Since Orion-RE uses a transformational system to recover an OO design with basis on the source code, there must exist transformers to recover design information directly from the code. However, these transformers must be first constructed, which requires great effort and knowledge about the involved languages. The time spent in the transformers construction is also a critical factor. As shown in Table 1, a great amount of time was spent in the construction of the Progress language grammar and the transformers of this case study. However, it must be stressed that this effort is later reused in any legacy system written in the same language. The time reduction obtained when using these transformers in the design recovery, as discussed earlier, also justifies this effort.

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Time spent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress grammar</td>
<td>320 hours (2 months)</td>
</tr>
<tr>
<td>ProgressKB (identify step)</td>
<td>600 hours (4 months)</td>
</tr>
<tr>
<td>ProgressKB ProgressOO (organize step)</td>
<td>500 hours (3 months)</td>
</tr>
<tr>
<td>ProgressMDLJava (recover step)</td>
<td>960 hours (6 months)</td>
</tr>
</tbody>
</table>

- **Technological restriction:** In order to fully take advantage of Orion-RE, the Software Engineer is restricted to some technologies, because MVCASE and JADE tools are fixed in some technology, in this case UML and Java. Other technologies can be used, however without the complete support from Orion-RE.

5. Summary

As mentioned before, although not originally idealized as such, Orion-RE environment implements most of the identified requirements for CBESRE:

- **Language and technology independence:** Although some tools, such as MVCASE and JADE, for example, are fixed in some technologies, Orion-RE can be employed to rebuild systems from any languages to any new languages or technologies;
- **Extensibility:** The support for new languages and technologies can be incorporated in Orion-RE by constructing the appropriate transformers;
- **Scalability:** Orion-RE can be used to rebuild the system in separated modules, which reduces the scope of the problem, facilitating the reengineering of large systems;
- **Support for Reengineering activities:** Orion-RE has support for the main reengineering activities, including: Information gathering, reverse and forward engineering;
- **Tool integration:** The common repository allows the data exchange between the tools of the environment, in a standard format. Although there is no workflow engine to coordinate the usage of the tools, the SRT offers well-defined steps, which the Software Engineer may follow to correctly use the tools of Orion-RE;
- **Reusability:** In Orion-RE, the reengineering product consists of more generic, distributed components, which provides a higher reusability degree than in conventional reengineering processes, where the products are mainly programs specific to some particular system;
- **Components Publishing:** The use of a central repository, to remotely store software artifacts, is used as the publishing mechanism for the components; and
- **Compositionality:** The Software Engineer, using the repository searching mechanism and the MVCASE tool, can recover and compose components into applications.
6. Related work

In [9], the authors present the Moose Reengineering Environment, a language-independent tool environment to reverse engineer. It consists of a repository to store models of software systems, and provides facilities to analyze, query and navigate through them. Models consist of entities representing software artifacts such as classes, methods, etc. The reverse engineering and the repository presented in Moose are also present in Orion-RE. Besides, Orion-RE covers all the reengineering life-cycle, not only the reverse engineering phase, as in Moose. Instead, a forward engineering phase is supported by Orion-RE, to re-implement systems using a component-based approach. This allows the Software Engineer to benefit from tool support when adding new functionalities to the system. Finally, Orion-RE provides a novel approach to automatically deploy distributed component-based systems, which facilitates testing and deployment.

In [50], the authors present the FermaT Workbench, an industrial-strength assembler re-engineering workbench consisting of a number of integrated tools for program comprehension, migration and re-engineering. The FermaT is capable of a much deeper semantic analysis of the assembler source code. A hard-to-understand assembler language program can be transformed into a more comprehensible high-level language program. Similar to Orion-RE, a step-by-step approach, first organizing the code, and then identifying higher-level elements, is used. This approach has the advantage to recover information from very low-level code, which is not achievable with Orion-RE. Instead, Orion-RE works in a more conceptual level, focusing in business-related information, which is not always clearly visible in the code. This ability to work in higher abstraction levels makes Orion-RE much more attractive to rebuild large commercial legacy systems.

In [5] an approach for database reengineering is presented. Based on the key aspect of iteration, aiming to reengineer few parts of the legacy system at a time, this approach eliminates all the ageing symptoms of the legacy data base. The new database can therefore be readily used to integrate data used by new functions introduced in the system. This allows the Software Engineer to benefit from tool support when adding new functionalities to the system. Finally, Orion-RE provides a novel approach to automatically deploy distributed component-based systems, which facilitates testing and deployment.

In [50], the authors present the FermaT Workbench, an industrial-strength assembler re-engineering workbench consisting of a number of integrated tools for program comprehension, migration and re-engineering. The FermaT is capable of a much deeper semantic analysis of the assembler source code. A hard-to-understand assembler language program can be transformed into a more comprehensible high-level language program. Similar to Orion-RE, a step-by-step approach, first organizing the code, and then identifying higher-level elements, is used. This approach has the advantage to recover information from very low-level code, which is not achievable with Orion-RE. Instead, Orion-RE works in a more conceptual level, focusing in business-related information, which is not always clearly visible in the code. This ability to work in higher abstraction levels makes Orion-RE much more attractive to rebuild large commercial legacy systems.

In [5] an approach for database reengineering is presented. Based on the key aspect of iteration, aiming to reengineer few parts of the legacy system at a time, this approach eliminates all the ageing symptoms of the legacy data base. The new database can therefore be readily used to integrate data used by new functions introduced in the legacy software. Moreover, the approach allows all the services offered by modern data base management systems to be exploited. Until this moment, Orion-RE has no support for database reengineering, focusing mainly on the behavioral and structural aspects of the legacy system. However, this task could be performed separately, by some other tool or manually, completing Orion-RE’s functionalities.

7. Conclusions

This paper presented a Component-Based Software Reengineering Environment. By integrating tools that were not originally designed to work together, with a reengineering process model (SRT) and a middleware platform, Orion-RE implements most of the identified requirements, covering a substantial part of the software reengineering activities. Different technologies, such as middleware, XMI, frameworks, design patterns and CBD principles, were used to construct tools with features that offer an efficient, comprehensive way to perform software reengineering. The use of distributed components makes this approach very attractive, since it allows the reconstructed systems to be distributed over a network. The use of software components allow the products of the reengineering to be reused in future developments, saving effort, cost and time.

One of the main contribution of this paper is that it presents a large case study, referring to a real application of Orion-RE in the industry. It allowed us to review its forces and weaknesses, leading to future improvements in the environment.

Future works shall deal with the extension of the Orion-RE environment to cover a wider range of the reengineering activities, such as data model restructuring and testing. Other works shall deal with active repositories mechanisms in the environment, to improve reuse. Currently, projects are being conducted aiming to accomplish adjustments and refinements in the environment. A download version is also being prepared, which will be soon available in our web site [42]. More details about this work may be found in the references.

8. Acknowledgements

The authors would like to thank the anonymous referees for their reviews and suggestions during this work. This work is supported by Fundação de Amparo à Pesquisa do Estado da Bahia (Fapesb)-Brazil, CAPES-Brazil and RHAECNPq-Brazil.

9. References


