An Approach to Software Architecture Recovery Aiming at Its Reuse in the Context of Domain Engineering

Aline Pires Vieira de Vasconcelos, Cláudia Maria Lima Werner (Advisor)
COPPE/UFRJ – System Engineering and Computer Science Program
P.O. Box 68511 – ZIP 21945-970 – Rio de Janeiro – RJ – Brazil
e-mail: [aline, werner]@cos.ufrj.br

Abstract

Domain Engineering (DE) represents an approach to software reuse that has been explored a lot in the past decades. It emphasizes the development of artifacts to a family of applications that share a set of common requirements. Although many DE methods have been proposed, there is still a need for a greater support in the DE processes, especially concerning the Domain Design phase. This phase mainly involves the creation of a Domain Reference Architecture (DSSA – Domain Specific Software Architecture) to a specific domain, which is a hard and costly activity. In this context, this paper presents a PhD research that involves the definition of a process to software architecture recovery and comparison aiming at the generation of artifacts to support a DSSA specification. The proposal presupposes the recovery of architectures from some legacy applications in a specific domain and their comparison, generating a list of commonalities and variabilities.

1. Problem Description

Domain Engineering represents an approach to software reuse that has been explored a lot in the last years. Its main goal is the construction of domain artifacts that can be reused in the development of applications in a given domain [1]. Many methods to support DE have been proposed (e.g. [2] [3] [4]). All of them agree that a DE process encompasses three generic phases, namely: Domain Analysis, Domain Design and Domain Implementation. Even though these three phases are prescribed, they only provide a sound support to the Analysis phase, leaving the other phases without an adequate support. This is the case for the Domain Design, an essential phase in the process, which establishes the infrastructure for components specification and integration.

The Domain Design phase is mainly concerned with the specification of a domain reference architecture. Some methods and tools have already been proposed to support a DSSA specification [4]. In [1] a tool is proposed to indicate a possible architectural pattern to a domain based on the prioritized quality attributes. In this approach, even though the system structure can be defined, there is no support to the specification of domain components and connectors. O’Brien and Stoermer [4], on the other hand, analyses legacy systems in a domain in order to find commonalities and variabilities among them, extracting the common parts to support a reference architecture specification. Although the extracted information can help in domain reference architecture and components specification, the authors do not define techniques and tools that effectively support architectures comparison. Moreover, in the proposed architecture recovery process, there is a lack of some decision making mechanisms that can help in architectural elements definition.
In this context, the work presented in this paper aims at the definition of a process to software architecture recovery and comparison in order to generate artifacts that can support a DSSA specification. As stated by Kang et al. [2], one of the most important sources of information for Domain Engineering is the set of legacy systems available in the domain. However, each of the phases prescribed in this PhD proposal is complex and require the development of appropriate solutions and tool support.

Software architecture recovery from legacy systems represents a reverse engineering activity that has been receiving a great attention from the software engineering community in the last years. It involves a set of methods for the extraction of architectural information from some lower level representations of a software system, such as the source code [5]. It intends to support program comprehension, software maintenance, software reengineering and software reuse. However, architecture recovery from legacy systems represents a complex problem because their architectural documentation frequently does not exist, and when it does, it is often out of synchronization with the implemented system [6]. In addition, as stated by Harris et al. [8], there is a long vocabulary distance between the architectural domain and the programming language domain, which increases the difficulty involved in the activity. Programming languages don’t have constructs that directly reflect architectural abstractions (e.g. constructor for a layer).

In order to provide solutions to these problems, architectural recovery methods, techniques and tools have been proposed [5] [6] [7] [8] [9]. They really offer contributions to the area, but some specific problems remain unsolved, such as: the recovery of architectures from systems that don’t have the source code available, a sound support for decision making in the architecture elements reconstruction, the detection of a larger set of architectural styles and patterns, the recovery of different architectural views and the reconstruction of architectures for systems in which many languages are employed. Some of these problems are treated by the proposal described in this paper.

After the architecture reconstruction, the second phase of the proposed approach deals with architectures comparison and a list of commonalities and variabilities generation. In this activity, many difficulties are also encountered. The recovered models tend to be large and complex, and techniques and tools that can help in their comparison are required. Besides, there is a difference in vocabularies employed by different systems which can make the comparison hard. Approaches, such as [4], detail these problems, presenting significant criteria to be considered in the comparison.

2. Theoretical Basis

2.1 Domain Engineering

As stated before, Domain Engineering involves the construction of artifacts for a family of applications or domain. In contrast to previous approaches to software reuse, which focused on the reuse of source code artifacts, DE involves the development of artifacts in different levels of abstraction, i.e. analysis, design and implementation, allowing the reuse to be achieved since the beginning of the software lifecycle. The goal of a DE process is the development of artifacts in a domain that can be reused by applications developed in this domain, in a process called Application Engineering (AE) [1]. While DE covers the development for reuse, the AE covers the development with reuse. These approaches define together a complete software reuse process.

DE processes encompasses three phases, namely: Domain Analysis, Domain Design and Domain Implementation. The Domain Analysis determines the set of common requirements in the application family, identifying reuse opportunities. Domain Design uses the analysis
results as input to specify and generalize solutions to the common set of requirements. Domain Implementation involves the transformation of the common requirements and design solutions in an implementation model which covers the identification, construction (or extraction) and maintenance of reusable components in the domain.

In this work, the focus is on the Domain Design phase, in which the DSSAs establish an essential role. They specify a structural organization for the application family, through a set of components and connectors, which is the basis for the implementation phase and for applications instantiation in the domain. According to Chung et al. [10], the software architecture, especially specified for reuse, is the artifact that effectively realizes, through the components relationships, the functional requirements of the domain and supports the quality attributes determined. More information about Domain Engineering can be found in [1] [2] [3].

2.2 Software Architecture

Software Architecture is an emergent discipline in software systems development. The architecture term represents both: a design activity and an artifact generated as a result of design. Shaw and Garlan [11] define software architecture as the description of elements from which systems are built, interactions among those elements, patterns that guide their composition and constraints on these patterns. In the software reuse area, architecture plays an important role, establishing the logical and physical structure in which components are connected. Approaches such as Domain Engineering emphasize the reuse through a reference architecture defined to a class of applications. Software architecture involves a rich body of knowledge covering many related concepts and issues, such as components, connectors, configurations, architectural description languages (ADLs), architectural styles and patterns, architectural view models etc. Architecture representation has been one of the most explored themes concerning software architecture. In order to represent software architectures, many approaches have been used. Architectural Description Languages (ADLs), Architectural Styles and Patterns and Architectural View Models are the ones that are explored in the work proposed in this paper. ADLs aim at formalizing software architecture representation, allowing architecture validation, simulation and transformation. Many ADLs have been proposed in the technical literature. However, because of their complexity, the use of ADLs is not already consolidated. Architectural styles and patterns characterize a family of systems in terms of their structural organization. Each architectural style or pattern prioritizes a set of quality attributes while not favoring others. One of the greatest advantages of architectural styles and patterns is the definition of a common vocabulary through which software architects can communicate their decisions. The architectural view models (e.g. [13]) emphasize the importance of describing architectures through different perspectives, involving static and dynamic aspects of the system. This assumption is adopted in the proposal presented in this paper. Information on software architecture can be found in [11] [12] [13].

2.3 Software Architecture Recovery

Software architecture recovery represents the reconstruction process of an architectural model from legacy systems, which encompasses: architectural elements (or components), their connections, and their types, according to the employed architectural styles or patterns. Moreover, architectural information must include a mapping from system functionality to architectural elements, showing how the functional requirements are realized in the system.
In order to reconstruct a system model that covers this architectural information, a well-defined process, some techniques and tools must be defined. The process for software architecture recovery is typically a two-fold one, involving an extraction phase and an analysis phase [5]. In order to support the extraction phase, static and dynamic reverse engineering techniques must be employed. This phase involves bottom-up activities.

In the scope of the analysis phase, which is a top-down one, some mappings of knowledge to the extracted entities and compositions occur. The composition activity leads to the clustering of recovered source code entities (e.g. classes, functions, variables etc.) in order to define the architectural elements. This composition step can be performed based on architectural styles and patterns semantics or based on domain or application knowledge, characterizing a pattern matching process.

Besides reverse engineering techniques and pattern matching, other techniques that have been used to support architecture recovery encompasses: program slicing, formal concept analysis, data mining etc.

Many approaches to software architecture recovery have been proposed. They can be classified in three categories, according to the source of knowledge prioritized in the process: architecture recovery based on architectural styles and patterns (e.g. [8]), architecture recovery based on domain patterns (e.g. [6] [7]), and architecture recovery based on application functional requirements (e.g. [9]). O’Brien et al. [17] propose another classification schema taking into account the level of automation in the process and the employed techniques.

3. Work Methodology and Current Stage

This PhD research is part of the Odyssey project, which involves the development of a reuse infrastructure based on domain models named OdysseyShare [1]. The OdysseyShare is being developed by the reuse group of the COPPE/UFRJ since 1998 and represents the infrastructure that will be the basis to support the proposed approach.

The proposed approach, as mentioned before, involves two phases: software architecture recovery and comparison. The work involves the recovery of architectures from some legacy applications in a domain and their comparison in order to generate a commonalities and variabilities report that can be used to support a DSSA specification. The target systems must be Object-Oriented and written in the Java programming language. To the software architecture recovery phase, a process has been defined. It is presented in figure 1.

The proposed architecture recovery is based on the mapping from functionalities (represented through use-cases) to source code entities. This mapping is achieved through the use-case modeling and dynamic reverse engineering activities. The dynamic reverse engineering encompasses application execution monitoring, events collection (tracing) and dynamic UML models representation associated to use-case scenarios. A dynamic analysis, which takes into account interaction patterns identification in the traces and the mapping from functionality to the traces collected, is the basis for the architectural elements reconstruction activity. Architectural elements reconstruction involves clustering source code entities into architectural abstractions. Besides the mentioned criteria, static and dynamic metrics for Object-Oriented applications will be used to contribute to the reconstruction process. The process is iterative and incremental and the architecture is reconstructed in cycles. Each cycle is driven by the use-cases modeled to that cycle.
Although the recovery process is driven by the functional requirements of the application, aiming at the aggregation of entities that implement coherent functionality in architectural elements, non-functional requirements also need to be characterized in the recovered architecture. In this respect, one has to identify some architectural and design patterns during the Patterns Detection activity, and to associate some quality attributes to them, according to the qualities they prioritize and their implementation by the target application. System experts and documentation must be accessed at this moment to discover the qualities that were originally implemented by the system and some architectural analysis method, such as SAAM (Software Architecture Analysis Method) [18], must be employed in order to certify that these qualities are really performed.

The second phase of the approach involves the comparison among the recovered architectures. To this end, XML diff algorithms will be investigated. The recovered architectures will be represented in UML and described in XML.

The methodology adopted to develop the proposed work involves a set of activities, namely: literature survey; proposed approach specification; study about software development environments and tools that support Domain Engineering and Software Architecture Recovery; tool support specification and development; qualifier monograph writing and presentation; papers writing; evaluation of the proposed approach through case studies; and thesis writing and presentation. Some of them are explained in more details as follows.

The literature survey covers the study on topics related to the proposed approach. Software Architecture was one of the first areas investigated. The investigation covered relevant subjects, such as: architectural styles and patterns, architectural description languages, architectural view models etc. Domain Engineering methods and their phases, mainly the Domain Design phase, were also studied. In this survey activity, the Reverse Engineering literature has been one of the most explored, involving a study on reverse engineering techniques, related to static and dynamic reverse engineering, and architecture recovery approaches. The literature survey started in April 2002 and will continue during the whole research period in order to follow the advances in the area.

The specification of the proposed approach has been done during the last months based on the literature review and on some tests performed over OdysseyShare source code and application. Through the recovery and analysis of traces related to some OdysseyShare use-case scenarios it was possible to detect, manually, interaction patterns and to verify that their participant object types in fact were part of the same element in the architecture. Actually, the
first phase has been more detailed than the second one, which will be further investigated and detailed during the next year.

The study of automated environments and tools to reverse engineering in general and architecture recovery in particular intends to allow an evaluation of their strengths and weaknesses in order to make clear where contributions can be given in this respect. As mentioned before, the infrastructure already selected to support the approach is the OdysseyShare since it covers the Domain Engineering and Application Engineering processes and offers some tools that properly fit to the proposal needs. Among the tools already offered by the Odyssey-SDE are: a tool for process modeling, execution and accompaniment; a tool for static reverse engineering from Java programs to UML class models; and a design pattern detection tool. For the activities to which OdysseyShare does not offer the necessary support, an appropriate tool support, which can fill in gaps found in the already existent tools, will be investigated and developed, as necessary.

It is important to emphasize, however, that the dynamic reverse engineering activity already has some tool support. In order to support this activity, many tools were investigated, but due to some specific requirements of the proposal, a tool has been developed in the context of the Odyssey project. The developed tool, the Tracer, uses Aspects [16] to monitor Java programs executions and to collect application execution traces. Aspects allow a non-intrusive system instrumentation and a selective collection in which packages and classes that aren’t of interest can be filtered. Moreover, through the Tracer it is possible to enable and disable the collection, making it possible to collect only the events related to a specific system functionality (e.g. a use-case scenario). A sequence diagram extractor module is also part of the tool and allows the extraction of sequence diagrams in the OdysseyShare related to use-case scenarios.

The activity related to the qualifier monograph writing and presentation was performed from November 2003 to May 2004.

Case studies to evaluate the proposed approach must be performed during the last year of the research period. The intention is to recover the architectures of some legacy applications in a domain (at least three) and compare their architectures by evaluating the results. Domain engineers, if available, and system experts can help in this evaluation, which intends to show if recovered architectures help in system comprehension and DSSA specification. Evaluation results will be used to refine the process.

The work methodology also covers papers writing and the thesis writing and presentation. Papers writing will occur along the whole research period in specific moments according to an agenda involving the conferences of interest. The thesis writing and presentation, the final activity in a PhD research, is planned to occur officially in the period from June 2005 to March 2006. However, the intention is to take advantage of the qualifier monograph and the papers written along the research period for the Thesis writing.

4. Related Work

Although software architecture recovery is a relatively recent area of research, many approaches have already been proposed. Kazman and Carrière [6] propose a software architecture recovery environment, named Dali workbench, which integrates many tools to support software architecture extraction, manipulation and analysis. The extracted model is stored in a relational database and represented in a graph in the Rigi tool [15]. In this work, the architecture is reconstructed through a pattern matching process, which involves writing queries over the recovered system graph and grouping the elements that compose architectural abstractions. The workbench specifies some tools required to architecture recovery and the pattern matching mechanism gives some power to model manipulation. However, only a
static logical view of the architecture is recovered and there is no support to decision making along the process.

Harris et al. [8] use architectural styles and patterns to guide the recovery. They developed some queries to recognize architectural styles on a C/Unix system. Although they give a great contribution to the area in the sense that they provide some automated support to architectural elements detection, their recognition mechanism is limited because it is based on programming language idioms found in C/Unix systems and on a static analysis of the software.

In [14], the recovery problem is treated as a pattern graph matching problem, in which the high-level design of a system is modeled as a pattern graph and matched with an entity-relationship graph representation of the source code system entities. In this matching process, data mining techniques are used. Although the approach gives some automation to the recovery process, the developer must have a good knowledge about the legacy system in order to model the high-level design.

Riva and Rodriguez [7] use a base of Prolog facts to represent the recovered model of the system and apply Prolog rules to cluster the entities into architectural components. The main contribution of the work is the recovery of architectural static and dynamic views and their integration in a modeling environment. Manipulations in one view, involving the aggregation or expansion of components, are reflected in the other.

O’Brien and Stoermer [4] select candidate systems in a domain, reconstruct their architectures and compare them in order to generate a report of commonalities and variabilities. Their goal is to create a Product Line from existing products, in case their commonalities justify this creation. The architecture reconstruction activity is performed using the Dali workbench [6]. In this work, many significant criteria and considerations to architectures comparison are given. Although it is a similar work in comparison to this Thesis proposal, the approach lacks the definition of some techniques and tools that can support the process. A lot of manual effort is required.

5. Expected Results

The following contributions are expected in this research:

- The definition of a software architecture recovery process in which the decision making support is based on the dynamic analysis of the system. Some process goals include a more sound support to decision making during elements composition and the recovery of static and dynamic architectural views based on the 4+1 view model [13].
- The detection of a large set of architectural styles and patterns in legacy systems through the dynamic analysis of the system.
- A support to software maintenance activities through the mapping from functionalities (use-cases) to system entities.
- A support to Domain Engineering in the DSSA specification activity through the recovered architectures and their list of commonalities and variabilities.
- A reduction of the manual effort required to software architecture recovery and comparison through the definition of some techniques and tool support integrated to a software development environment.

6. References

Component-Based Development”, In: Information Reuse and Integration (IRI), Las Vegas, Nevada, USA, October 2003.


