Framework extraction with domain analysis

GIANCARLO SUCCI[‡], ANDREA VALERIO[£], TULLIO VERNAZZA[§], MASSIMO FENAROLI[f], PAOLO PREDONZANI[§]

[§] *Dipartimento di Informatica, Sistemistica e Telematica - Università di Genova*
[‡] *Department of Electrical and Computer Engineering - The University of Alberta*
[f] *Thera S.p.A., Brescia, Italy*
[£] *Coclea, Rovereto, Italy*

Sherlock is a domain analysis methodology for the extraction of reusable frameworks. Sherlock is based both on Proteus and on FODA domain analysis techniques. The input of Sherlock is a description of the domain based on domain applications, literature, user requirements, and interviews with domain and market experts. The output of Sherlock is a framework comprising a set of components organized in architectures. In this paper we describe the main features of Sherlock and present a case study regarding the development of a graphic user interface framework for business and management information systems.

1 INTRODUCTION

Systematic software reuse improves the quality and the productivity of a software firm [1]. Systematic reuse requires an understanding of previous work. A major problem is the creation of assets that can be reused in a context different from those where they have been used previously. Domain analysis processes existing and potential software applications to extract and pack reusable assets. In this view, domain analysis is a fundamental activity for systematic software reuse.

Several approaches to domain analysis exist. However, they are not widespread. One reason could be that sometimes they are too difficult and rigid; another reason could be that they target assets that do not have high reuse potential. Almost none of them are specifically targeted to design frameworks. In this paper we outline how we used a domain analysis technique – Sherlock – to extract reusable frameworks. Section 2 defines the domain analysis process and outlines the main features of Sherlock. Section 3 details the case study. Section 4 draws the conclusions.

2 DOMAIN ANALYSIS AND SHERLOCK

Domain analysis can be defined as “a process by which information used in developing software systems is identified, captured and organized with the purpose of making it reusable when creating new systems” [2].

In these last years many authors have proposed different approaches to domain analysis [3]. Table 1 gives a short summary of them.

[‡] Corresponding Address: Giancarlo Succi, 238 Civil/Electrical Building, The University of Alberta, Edmonton, AB, T6G 2G7, Canada. Email: giancarlo.succi@ee.ualberta.ca

These methodologies are not widespread. Most of them do not aim to generate reusable frameworks. Although these methodologies are enabling factors for reuse practices, they do not provide the practical tools to achieve them. As a result, a great effort is necessary to link theory to practical results.

Another shortcoming of these methodologies is that they do not address whole domains – as sets of several applications – but only single applications. Single applications are the object of traditional application analysis, which is a very restrictive with respect to the larger scope of domain analysis. Some methodologies consider domains of several applications when explaining the theory, but provide examples only for single applications.

Proteus and FODA have had a significant influence on Sherlock. Sherlock takes from Proteus the general process based on object oriented techniques, and from FODA the domain characterization activity. Yet, Sherlock addresses the weaknesses of these and of the other methodologies. The strong points of Sherlock are the following:

- The methodology generates frameworks, an asset that has a direct impact on reuse.
- The methodology has an integrated approach, clearly stating the goal – software reuse – and showing where to take the information from and how to use it to achieve the goal. The clear definition of all the procedures allows systematic and planned reuse.
- Domain characterization is intensified to

Table 1: Main characteristics of some of the domain analysis methodologies proposed in literature.

Methodology	**Characteristics**
Product-oriented paradigm [4]	Domain characterization (called “market analysis”) is explicitly prescribed. Domain models are not defined, but the collection of reusable entities, non-entities and abstractions could fill this role.
Domain analysis for reusability [5]	The process proposed prescribes bottom-up classification activities and top-down system analysis activities. The outcome is a collection of reusable components organized using faceted classification. Planning is emphasized.
Feature-oriented domain analysis [6]	FODA is focused on the extraction of features (prominent, user-visible aspect) of a system, classifiable into functional, operational and presentation features. FODA uses established modeling techniques, such as entity-relationships diagrams, emphasizing a functional view of domain applications.
Domain analysis in the Synthesis environment [7]	This methodology is similar to FODA and refers to the DOD 2167A standard process. It has a preparation phase dived into domain description and domain qualification. Domain qualification focuses on feasibility and costs.
Domain analysis in RAPID [8]	RAPID methodology (for ADA) aims to facilitate reusability. Data collected about design of existing systems and interviews with experts are archived in a Domain Information Database.
Domain analysis for building an organon [9]	The purpose is to incrementally build “intelligent” libraries embedding knowledge-based reasoning facilities that support taxonomic reasoning. Domain analysis is seen as an evolving process.
Domain analysis in IdeA [10]	This methodology aims to reuse abstract software designs, with problem-solving operators and solution steps. IdeA is cyclic and close to the evolutionary nature of domain analysis.
PROTEUS domain analysis methodology [11]	PROTEUS identifies the common and invariant parts of a system in the domain. The emphasis is on supporting evolutionary software development. Using object-oriented techniques, it supports the identification and characterization of domain variability. This methodology prescribes explicit documentation of domain models and model validation.

perform a thorough investigation of the domain/market taking into account all the variation points that significantly affect the applications.

- The methodology uses UML [12] as the modeling language, enriching it with the semantics of domain analysis.

Sherlock's activities are *definition of the context, domain characterization,* and *framework development*.

In the *definition of the context* Sherlock defines the purpose of the domain analysis and the boundaries of the domain under analysis.

In *domain characterization* Sherlock collects relevant information in the domain taking into account past, present, and potential future applications. It identifies and formalizes commonalties and variabilities of the domain. Commonalties are the set of features that are common to several applications. Variabilities are what makes applications different from each other. Sherlock localizes variability in variation points. Each variation point can be implemented through several variants. A given choice of variants determines an application. Market-driven variation points divide the domain/market in partitions. A firm positions its applications and does strategic planning on this partitioning.

In *framework development* Sherlock implements software components to support the development of the applications in the domain and defines architectures to structure those components. Frameworks embody solutions to common domain issues and solutions and mechanisms for dealing with variability in application specific issues. The structuring software reflects the structuring of the domain in domain patterns.

A full description of Sherlock is in [13].

3 THE APPLICATION OF SHERLOCK IN THERA

The case study presented regards the development of a Graphic User Interface (GUI) framework for business and management information systems. The project was conducted during the first 5 months of 1997 inside Thera S.p.A., one of the major Italian firms in information system production for private and public bodies. The required effort was 20 person-months. The GUI framework delivered at the end of the project is currently used inside different software systems.

3.1 *Definition of the context*

The purpose of the analysis is to provide a GUI framework to display Business Objects belonging to a Business Model (Figure 1). Business Objects represent customers, sales, agents, transactions, etc. The framework should be flexible enough to handle different user requirements and different types of Business Objects.

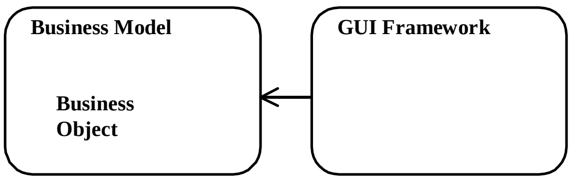

Figure 1: Context definition

The framework should assume that the Business Objects are already present in existing applications, so it should not put constraints that require changes to the Business Objects. The framework should be able to display from different viewpoints and to manipulate graphically the Business Objects.

The framework should include high level, business-oriented components, and should leave out low level GUI components, which are available from third parties.

3.2 *Domain characterization*

We collected information about the market and technical information.

The information about the market was obtained from interviews with market experts and users. These gave a high level characterization of the applications in the domain without considering their technical aspects. This information also provided a clear view of what part of the market the firm was addressing and what were the strategic moves for the future. We identified two significant variation points: the user of the application and the application area. The 'user' variation point has the following variants: operational level, middle management, and top management. The 'application area' has the following variants: services, banking, and commerce/production. Figure 2 shows a partitioning of the domain/market in nine partitions according to the variants of the variation points.

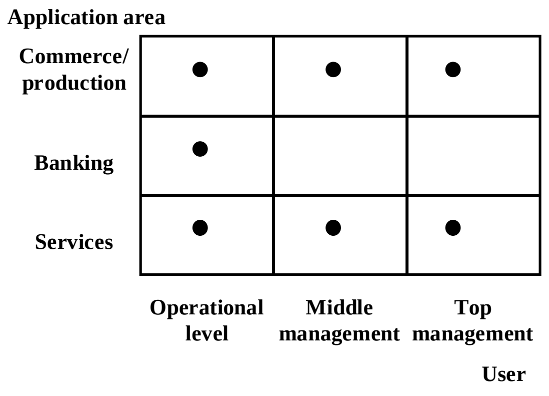

Figure 2: Domain/market partitioning

The spots in the partitioning represent application the firm already has. For the purpose of domain analysis, requirements of all the available applications should be considered. Moreover potential application the firm is interested in should be considered too. Figure 3 shows two applications (in gray) the firm could possibly develop. The applications regard banking systems GUIs for middle and top management. The firm has decided to move to those applications through expansion of the existing operational level banking application.

The technical information we have identified and classified concerns past project documents, manuals, standards, and literature dealing with the GUI domain.

The common part of all the applications comprises the following operations: display, insertion, modification, browsing, querying. Yet, these operations need different implementations in different applications. We capture the differences in a set of variation points:

- Graphical look and feel: users have different expectations about the GUI appearance and require interaction methods that fit their particular tasks and goals.
- Information filters: although different GUI applications access the same data of the Business Model, not all the information must be presented. Filters ensure that data are sorted and presented to the applications that need them.
- Runtime customization: this handles preferences and favorite information sets the user can specify to customize his or her GUI environment runtime.
- Underlying Business Objects: several Business Objects of different type need to be displayed through the GUI.

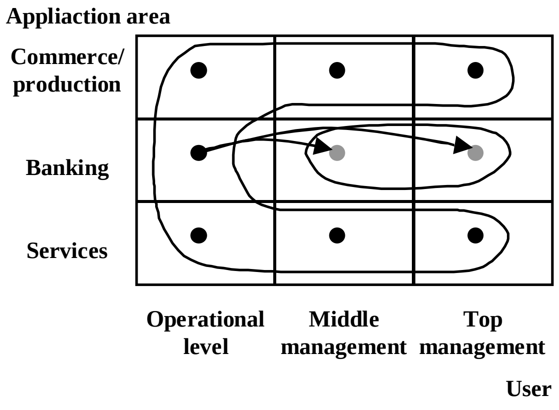

Figure 3: Market plan on the partitioning

Each of these variation points has a set of variants that implement them and specify the undetermined parts. Supporting variation points in the framework through components that address single variants, we can build applications as custom assemblies of components.

3.3 Framework development

Frameworks comprise components and architectures. In a OO approach [14] components are C++ classes. Architectures are necessary to give a logical structure to the components. In Sherlock there are three architectures: functional, static and dynamic.

The **functional architecture** specifies what services are available. These are **Display, Action,** and **Control.** Display manages the display of Business Objects, supports information filtering and runtime customization, and controls the uploading of the information from the database. Action controls the UI objects, their connections, and the dispatching of events; it is also responsible for the management of errors. Control links the GUI to the Business Model and ensures consistency between them.

The **static architecture** shows the interfaces/classes and the relations between them. The high level interfaces/classes are the following (Figure 4): **Container Window** (CW), **Setting Window** (SW), **Object Editing Window** (OEW), **Search Window** (RW), **Key Window** (KW). CW presents the details of the instances of a Business Object that comply with given filters. SW is the configuration panel for the customization of the Container Window. OEW is the window that guides the maintenance of a Business Object. RW is a Container Window that displays the instances found with a query on the code or on the description. KW shows the identification of a Business Object.

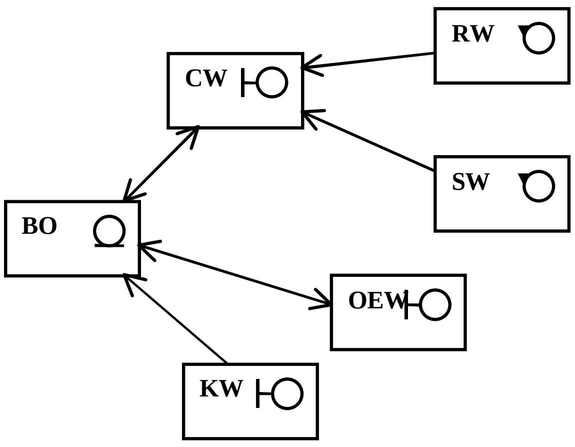

Figure 4: Static architecture

The **dynamic architecture** presents a dynamic description of the behavior of the system in terms of objects and interactions between objects. A snapshot view of the communications not taking into account time is shown in the object dynamic architecture (Figure 5).

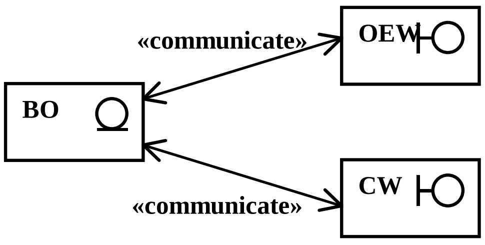

Figure 5: Object dynamic architecture

The object dynamic architecture is different from the static architecture: the behavior of a system may emerge from the communication of several objects of the same class; this can be represented in the object architecture but not in the static architecture. An interaction dynamic architecture represents communications between objects versus time (Figure 6).

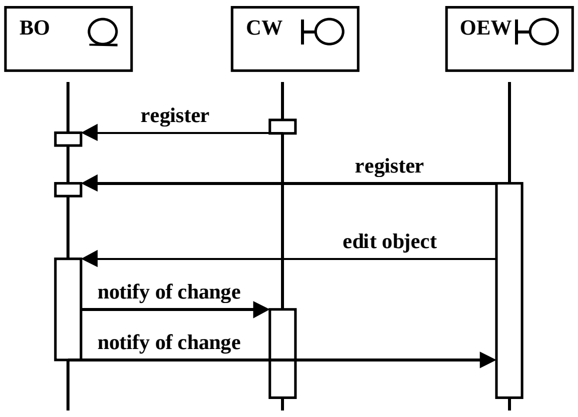

Figure 6: Interaction dynamic architecture

We attached a textual description to each diagram of the dynamic architecture. The descriptions are frequently applications of known design patterns [15].

4 CONCLUSIONS AND FURTHER RESEARCH

Frameworks are good candidates for software reuse. A domain analysis methodology targeted to frameworks such as Sherlock is the enabler to framework reuse. This work presents a case study of the application of Sherlock in Thera

S.p.A. Several working units are now reusing the resulting reusable framework; therefore, the overall result of this experiment is excellent.

However, a lot of work has still to be done. The gains should be quantified. The methodology should be stabilized. The authors aim to address these issues in the next few years.

ACKNOWLEDGMENTS

This work has been partially supported by the European Commission under the framework of the Esprit Project Seldom.

REFERENCES

1. W. C. Lim, *Effects of reuse on quality, productivity and economics*, IEEE Software, 11(5), 1994, pp. 23-30.
2. R. Prieto-Diaz, *Domain Analysis: an Introduction*, ACM SIGSOFT - Software Engineering Notes, vol. 15, no. 2, April 1990, pp. 47-54.
3. G. Arango, *Domain Analysis Methods*, Software Reusability, W. Schaeffer, R. Prieto-Diaz, M. Matsumoto, Ellis Horwood, New York, 1993.
4. R. McCain, *Reusable Software Components Construction: a Product Oriented Paradigm*, Proceedings of the Fifth AIAA/ACM/NASA/IEEE Computers in Aerospace, 1985.
5. R. Prieto-Diaz, *Domain Analysis for Reusability*, Proceedings of COMPSAC 87: The Eleventh Annual Intenational Computer Software and Applications Conference, IEEE Computer Society, Washington DC, October 1987.
6. J. Hess, S. Cohen, K. Kang, S. Peterson, W. Novak, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Software Engineering Institute, 1990.
7. G. Campbell, S. Faulk, D. Weiss, *Introduction to Synthesis*, Techincal Report Intro_Synthesis-90019-N, v. 01.00.01, June 1990, Software Productivity Consortium, Herndon, VA 22070.
8. W. Vitaletti, E. Guerrieri, *Domain Analysis within the ISEC Rapid Center*, Proceedings of the Eighth Annual National Conference on Ada Technology, March 1990.
9. M. Simos, *The growing of an Organon: a hybrid knoledge-base technology and methodology for software reuse*, Domain Analysis and Software Systems Modeling, R. Prieto-Diaz and G. Arango Editors, IEEE Computer Society Press, Los Alamitos, CA, 1991.
10. M. Lubars, *Domain Analysis and Domain Engineering in IDeA*, Domain Analysis and Software Systems Modeling, R. Prieto-Diaz and G. Arango Editors, IEEE Computer Society Press, Los Alamitos, CA, 1991.
11. Hewlett Packard, Matra Marconi Space, CAP Gemini Innovation, *Domain Analysis Method*, Deliveable D3.2B, PROTEUS ESPRIT project 6086, 1994.
12. The Rational Software Corporation, *The UML Document Set*, URL: http://www.rational.com/uml
13. P. Predonzani, G. Succi, A. Valerio, *ENEL-SERN-TR-98001*, Department of Electrical and Computer Engineering, the University of Calgary, 1998.
14. M. Fayad, D. Schmidt, *Object-Oriented Application Frameworks*, CACM, Vol. 40, No. 10, October 1997.
15. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.