Information and Software Technology 51 (2009) 448–459

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

CompAS: A new approach to commonality and variability analysis with applications in computer assisted orthopaedic surgery

Gisèle Douta [a,*], Haydar Talib [a], Oscar Nierstrasz [b,1], Frank Langlotz [a]

[a] MEM Research Center for Orthopaedic Surgery, Institute for Surgical Technology and Biomechanics, University of Bern, Stauffacherstrasse 78, CH-3014 Bern, Switzerland
[b] Software Composition Group, Institute of Computer Science, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland

ARTICLE INFO

Article history:
Received 7 November 2006
Received in revised form 21 August 2007
Accepted 26 May 2008
Available online 12 June 2008

Keywords:
Computer-assisted surgery
Software reuse
Component-based programming
Domain analysis
Commonality and variability
Software evolution

ABSTRACT

In rapidly evolving domains such as Computer Assisted Orthopaedic Surgery (CAOS) emphasis is often put first on innovation and new functionality, rather than in developing the common infrastructure needed to support integration and reuse of these innovations. In fact, developing such an infrastructure is often considered to be a high-risk venture given the volatility of such a domain. We present CompAS, a method that exploits the very evolution of innovations in the domain to carry out the necessary quantitative and qualitative commonality and variability analysis, especially in the case of scarce system documentation. We show how our technique applies to the CAOS domain by using conference proceedings as a key source of information about the evolution of features in CAOS systems over a period of several years. We detect and classify evolution patterns to determine functional commonality and variability. We also identify non-functional requirements to help capture domain variability. We have validated our approach by evaluating the degree to which representative test systems can be covered by the common and variable features produced by our analysis.

1. Introduction

Computer-assisted orthopaedic surgery (CAOS) is a technological domain that arose in the early 1990s from the combination of several other mature sciences such as image processing, biomechanics, computer graphics, and robotics. Orthopaedic surgical procedures follow the common basic principle of "placing an object (guide wire, screw, tube or scope) at a specific site, via a trajectory which is planned from medical images and governed by three-dimensional anatomical constraints" [1]. In order to provide surgeons with a means to perform these procedures with higher accuracy, CAOS systems have been progressively introduced into the operating room. Using virtual representations of the surgical instruments and of the operated anatomy, CAOS systems replay in real time the surgeon's actions on a computer screen (Fig. 1). Although many technical approaches have been taken to develop these systems their conceptual designs remain similar: CAOS systems typically consist of a planning subsystem to help the surgeon define the optimal surgical strategy and a navigation subsystem to support him or her in achieving the planned strategy [2,3].

Because of the commonality in surgical gestures the variety of CAOS systems developed to assist in diverse orthopaedic surgeries offer common features such as loading/acquisition of medical data, data visualization in 2D and/or 3D, and selection of the best fitting implant. However, up to now each application is considered as an individual system strictly bound to a specific surgical procedure and pathology. Such a system engineering approach results in monolithic systems that do not have the flexibility required to allow one to take advantage of the functional similarities of these systems through software reuse.

The basic idea underlying software reuse is simple: rather than building software systems from scratch we assemble them from common reusable assets such as modules, objects and classes. Component-based programming is a recently-established paradigm for software reuse. According to Szyperski [4] "a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"'. In other words components are the building blocks from which an application can be composed in a "plug and play" manner. Adopting such a software development approach implies a move from single systems engineering to families of systems. A system family is a set of software applications sharing a large number of common properties [5]. Domain engineering refers to methods for defining, designing and implementing the necessary assets to support software reuse in system families. The initial

* Corresponding author. Tel.: +41 31 631 5959; fax: +41 31 631 5960.
E-mail addresses: Gisele.Douta@MEMcenter.unibe.ch, douta_gisele@yahoo.fr (G. Douta).
[1] Tel.: +41 31 631 4692; fax: +41 31 631 3355.

0950-5849/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2008.05.017

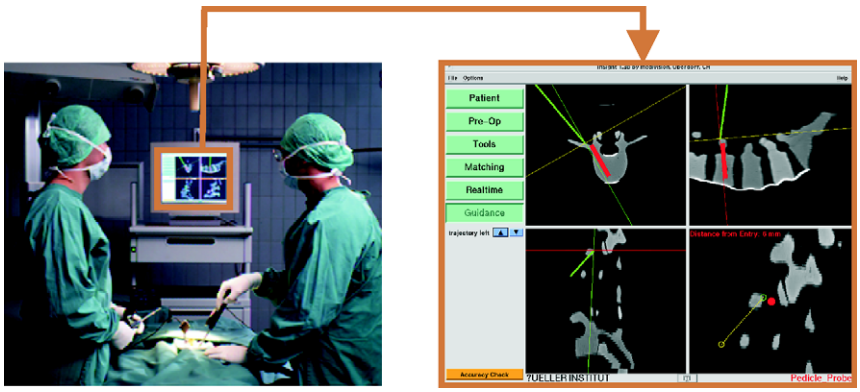

Fig. 1. Computer-assisted orthopaedic navigation.

and crucial step of these software engineering methodologies is called domain analysis. It aims at identifying commonalities, variabilities and dependencies in the selected family and at integrating them in a coherent model [6].

In order to take advantage of the functional similarities present in the CAOS family of applications, we propose to apply component-based programming to the development of CAOS systems. Because it is the necessary prerequisite to enable efficient component-based software reuse we focused first on domain analysis. We have designed CompAS, a new approach to commonality and variability analysis to support component-based architectural modeling. The key novelty of our approach is to analyze the evolution of the domain to effectively determine which features should be included as common or variable.

2. Challenges in performing domain analysis in CAOS

A domain model is the set of artifacts resulting from the domain analysis. The appropriate domain model is the one that provides the most sensible system decomposition in terms of common and variation points. Its achievement requires a careful balance between current and future needs. This information can usually be extracted from interviews with domain experts, existing systems, and literature. Yet the software development context considered here is a research environment where, contrarily to the usual industrial approach, the most common practice is to implement prototype applications more or less from scratch, in order to allow the clinical validation of the investigated concepts, which usually implies inconsistent system implementation documentation. Moreover, among the potential candidates for the investigated family of applications only a restricted number of them were implemented at our institute. This means that we had access to the code of only few of our application family's members. However, CAOS has the particularity to be a domain for which research and industry are still not only continuously innovating but as well publishing these innovations. We propose a method that takes advantage of this extended and publicly available literature to palliate our lack of systems documentation.

The identification of commonalities and variabilities mainly relies on the capabilities of the domain analyst to abstract from and refine the collected data and knowledge. We propose to strengthen the process of commonalities and variabilities identification with a quantitative evaluation of functional evolutionary trends. Several methodologies have been proposed to evaluate software evolution, one of the main differences between them being the type of data they require as input. Some methods extract evolution trends from version control data such as that provided by the Concurrent Version System CVS [7,8]. This information (e.g. modification reports), can be combined with problem reports extracted from a bug tracking system and with feature information derived from the executable itself to visualize feature evolution [9]. In our case where only scarce source code data are available we were inspired by the telephony feature evolution study performed by Anton et al based on publicly available information about telephony [10] to consider literature as our data source. We suggest using evolution matrices, which track the evolution of features over time, to expose implicit patterns in natural lifecycle of features [11].

Software family engineering not only focuses on currently existing systems but it anticipates future needs and variations as well. The results of the domain analysis must then appropriately model variations so that it provides:

- the software user with an explicit and concise representation of available variabilities;
- the developer of reusable software with the knowledge why a certain variation point is included in the software;
- the software architect with the basis to design an architecture flexible enough to support the family diversification and evolution.

Apart from the desire of continuously proposing more appropriate and useful functionalities, CAOS research also aims at providing innovative methods and technology to implement these functionalities. In order to model CAOS variability at the functional and technological level we propose a taxonomy of change scenarios. By taxonomy we simply mean the dictionary definition of "a system for naming and organizing things ... into groups, which share similar qualities" [12]. By change scenarios we refer to situations, where only a particular functional or technological aspect of an existing system is modified.

3. Domain analysis

Domain analysis is the step of domain engineering during which the domain analyst selects a family of applications (or domain) to study, collects the domain knowledge, organizes it into a set or artifacts (domain models) describing the common and variable properties of the system family, and defines the semantics of these properties and the dependencies between them. A large number of domain analysis methods exist and all of them agree that the appropriate source of information should mainly come from [6,13,14]:

- human sources: domain experts, system users, developers, etc.;
- existing systems: source code, design documentation, user manuals, etc.;