TracED: A tool for capturing and tracing engineering design processes

María Luciana Roldán *, Silvio Gonnet, Horacio Leone
CIDISI – INGAR/UTN – CONICET, Avellaneda 3657, 3000 Santa Fe, Argentina

A R T I C L E   I N F O
Article history:
Received 14 November 2008
Received in revised form 25 June 2010
Accepted 28 June 2010
Available online 31 July 2010

Keywords:
Design process support
Design history
Software architectures
Knowledge acquisition
Architectural design decisions
Operational approach

A B S T R A C T
The design of products or production processes in many engineering disciplines such as chemical, or software engineering, involves many creative and sometimes unstructured activities, with an opportunistic control flow. During these design processes, several models are generated, which have different levels of abstraction of the object being designed. Given the difficulties in dealing with this complexity using an improvised way, there is an urgent need for tools that support the capture and tracing of this process. In this proposal, TracED, a computational environment to support the capture and tracing of engineering design process is presented. It allows defining a particular engineering design domain and supporting the capture of how products under development are transformed along an engineering design process. Particularly, in this work, we consider software architectures design domain. As in any complex process, the support of computational tools is required for enabling its capture.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

Product and production process design in many engineering disciplines such as chemical, or software engineering is a challenging task. Over the last 30 years, the engineering design process has been transformed by the introduction of massive computing power, where problem-solving environments (PSEs) play a main role. PSEs aim to assist engineers in solving several complex tasks.

A design PSE must be able to handle all requirements of a design team and to integrate with the stages of the design process. Moreover, it must be configurable by the design team itself to suit each new problem as it is tackled [1]. The design PSE should be flexible and should not constrain the task of the designer in an unnatural manner [2]. In order to assist engineers during the design process, it is necessary to understand such a process and to have a computer representation of it. However, design problems are ill-structured, because the designer does never have enough information in the initial state and the properties of the goal state are never fully specified in advance. Therefore, many different goal states are conceivable and acceptable [3,4]. All this become designing in a nondeterministic process, which is difficult to model and even more difficult to prescribe [5]. As a consequence of the previously pointed out features, there is a real need of supporting tools that could capture how an engineering design process was carried out. By having such tools, the tracking and tracing of the design process would be possible, as well as the analysis of its rationale.

In this way, the design experts’ knowledge could be captured, thus providing the foundations for learning and training activities and future reuse.

Several attempts to provide support to the design process in different engineering domains have been reported [6–8]. Some tools are based on design reasoning capture by means of the concepts proposed by the IBIS model [9]. Another line of research related to design is the management of development processes products (where products are models, data, diagrams, etc.). For some time now, there have been widely used systems for managing products and their versions [10]. This practice responds to the basic need of storing and organizing the products of a development process [11]. These management systems, like software configurations-managing systems, are focused on products, and they do not consider the design process tracing. Consequently, these tools do not satisfy the need of capturing the design process together with its reasoning.

In this paper, TracED, a computational environment to support the capture and tracing of engineering design process is proposed. The environment is based on a generic model for capturing the design process in terms of the operations applied to the design objects [12]. Its goal is the capture of the developed model versions during a design process; in other words, it represents the design states and how they were obtained. The environment was designed with the requirement of supporting various design domains, therefore, TracED could be adapted to each new design problem, according to the particular concepts of a given design domain and the possible operations that can be applied over the instances of those concepts. For example, regarding software engineering, one of the most important stages is the software architecture design process
provide such features, all the information involved and used in the design process should be formalized and modelled in a suitable format for automatic data processing. Therefore, the scheme to capture and trace an engineering design process proposed in this paper is a mixed approach that combines object-oriented technology and situational calculus [23,24]. This choice is justified by the following reasons: (i) object-oriented approaches make the knowledge representation task much simpler because they reflect a more natural view of the domain to be modelled, and the model extension requires no strategic changes in the structure of the knowledge base itself; (ii) by means of the situation calculus, the evolution of the products of a design project is represented. The situation calculus is a first-order language for representing changes, sometimes enriched with some second-order features [24]. The basic concepts are situations, actions, and fluents. Briefly, actions are what make the dynamic world change from one situation to another. Fluents are situation-dependent functions used to describe the effects of actions. Possible world histories, which are sequences of actions, are represented by first order terms called situations. Situation calculus is perfectly suitable to model dynamic worlds, which, in this case, is the evolution of a model in an engineering design process. This evolution is specified by using successor state axioms, first order formulas that encode the axioms about how the products of a design project evolve. Therefore, situation calculus is used to formalize the evolution of a design process and object-oriented technology to represent the main concepts to enable the construction of computational tools that implement the formulated specification.

The proposed scheme considers the design process as a sequence of activities that operate on the products of the design process, named design objects. Typical design objects are models of the artefact being designed (i.e. an information system, an industrial piece of equipment, or a chemical process plant), specifications to be met (i.e. stream purity specs, products’ throughput for process system engineering, quality attributes such as modifiability or performance for software engineering), variable values (i.e. reflux ratios, number of stages of a separation unit, operating temperatures and pressures, etc.). Naturally, these objects evolve as the design process takes place, giving rise to several versions that must be kept. These versions may be considered as snapshots taken to design objects at a given point of time; and the set of those versions conform a model version. A model version describes the state of the design process in that time, including the artefact being designed. For example, Fig. 1 partially shows two model versions, $m_1$ and $m_2$. Both model versions are the result of a fragment of a SADP and include the structure of the artefact being designed. The example corresponds to the case study described in Section 3.

In this scheme, each model version is generated by applying a sequence of operations on a predecessor model version. The sequence of operations may include the elimination, creation, and modification of versions that constitute the predecessor model version. As it is exemplified in Fig. 1, the model version $m_i$ is obtained from the model version $m_0$ by deleting the WebApplication component and adding the components View, Model, and Controller, with their ports and connectors. Therefore, all model versions may have zero or more successor model versions and must have only one predecessor model version (except for the initial model version, which does not have a predecessor model version). Consequently, the representation scheme of versions is a tree structure, where each model version is a node and the root is the initial model version. This tree structure is illustrated in Fig. 1. There, it is possible to see the several successor model versions that have been proposed (Model Version $m_0$, Model Version $m_1$) from the initial model version (Model Version $m_0$), by applying the sequence of operations $\phi$ and $\phi_k$ respectively. This model evolution is posed as a history made up of discrete situations. The situation calculus [23,24] is adopted.
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات