Scenario Analysis in an Automated Tool for Requirements Engineering

Hong Zhu\textsuperscript{a} and Lingzi Jin\textsuperscript{b}
\textsuperscript{a}School of Computing and Mathematical Sciences, Oxford Brookes University, Oxford, UK; \textsuperscript{b}Cherwell Scientific Ltd, Oxford, UK

A scenario is a set of situations of common characteristics that might reasonably occur in the use of a system. It is widely recognised that scenario analysis can play a significant role in requirements acquisition, modelling and analysis. However, for a complicated software system, there may be a great number of scenarios. Synthesising a requirements definition from a set of scenarios and managing the consistency and the completeness of a set of scenarios are difficult and demand automated tool support.

This paper presents an automated tool for scenario-driven requirements engineering where scenario analysis plays the central role. It is shown that a scenario can be described by three views of data flow, entity relationship and state transition models by slight extensions of classic data flow, entity relationship and state transition diagrams. The notions of consistency and completeness of a set of scenarios are formally defined in graph theory terminology and automatically checked by the tool. The tool supports automatic validation of requirements definitions by analysing the consistency between a set of scenarios and requirements models. It also supports automatic synthesis of requirements models from a set of scenarios. Its utility and usefulness are demonstrated by a non-trivial example in the paper. Case studies of the tools are also presented.

\textbf{Keywords:} Automated tool; Completeness; Consistency; Requirements engineering; Requirements model synthesis; Scenario analysis

1. Motivation

Requirements analysis and specification are concerned with eliciting, clarifying and documenting users’ requirements of computation systems [1]. Many studies have shown that errors made at this stage are very costly, even impossible, to rectify. Neglected or only partially completed requirements analysis tends to lead to problems later in development. It is perceived as an area of growing importance. In the analysis and specification of users’ requirements, software engineers are often confronted with difficulties due to the complexity of the problem, the communication barriers between people of diverse backgrounds, the inconsistency and incompleteness of information and frequent changes of users’ requirements. A number of proposals have been advanced in the literature to overcome these difficulties. These include the deployment of various vehicles for separation of concerns, design and use of various kinds of representations and notations for the description of users’ requirements, and the development of software tools to support the requirements engineering process.

1.1. The Notion of Scenario and its Role in Requirements Analysis

A vehicle for separation of concerns is the concepts of scenarios and use cases [2–4]. In this paper, we define the notion of scenario as a set of situations of common characteristics that might reasonably occur in the use of a system. Although situations of use are normally only related to a part of the system’s functions, not every arbitrary subset of functions constitutes a meaningful scenario. A scenario should have the following characteristics.
User agents: a scenario must have a specific type of user, or a set of types of users, that participates in the use of the system in the scenario, and/or a subset of the equipment in the environment system that is involved in the operation of the system.

Use purpose: a scenario is the set of situations when the user or users use the system with a specific goal or purpose.

Operation condition: a scenario must occur under certain operational conditions and certain states of the environment system.

For example, consider a personal bank account management system. An example of a scenario is that a customer wants to withdraw some money from his account. Here, a customer of the bank is an agent. Other types of users of the system can be managers of bank branches and clerks of the bank. In this scenario, the goal or purpose of using the system is to withdraw money from an account. Other purposes can be to deposit, to make an enquiry about the balance of the account, or even to try to get money from another person’s account illegally. The condition under which a customer uses the system also varies. For example, a customer may try to withdraw an amount of money in excess of the balance in the account, or want to withdraw less than the balance, etc.

By identifying a set of scenarios, i.e. a set of such situations of common characteristics, and working on each scenario separately to elicit information, a complicated problem can be decomposed systematically and naturally. Information from different sources can be elicited independently. A requirements definition can be synthesised from the descriptions of the scenarios. Secondly, by analysing the consistency between requirements of different scenarios, conflicts between different types of users, different use purposes and different operation conditions can be discovered. Moreover, scenario analysis also provides a basis for requirements validation. By analysing the consistency and completeness of the requirements definition with respect to a set of well-defined scenarios, the requirements definition can be validated and verified.

A notion closely related to scenario is use case. Although there is no widely accepted definition of the terms ‘scenario’ or ‘use case’, many authors consider scenarios as instances of use cases, while many others use them as interchangeable. For example, in UML [5], a use case is defined as ‘the specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors’. UML’s definition of the notion of scenario is ‘a sequence of actions that illustrates behaviour’ and it ‘may be used to illustrate an interaction or the execution of a use case instance’. In other words, within UML, scenario refers to a single path through a use case. In this paper, instead of requiring a scenario to be a linear sequence of interactions, we consider a scenario as a set of situations that may occur in the uses of a system, and hence a scenario may be non-linear. Moreover, we will work towards a more general theory about the relationships between such sets of situations and calculations on them, such as the relationships that a situation is a sub-situation of another and two situations are consistent, and the operation that combines a set of situations into a model. Therefore, we will use scenario as a synonym for use case in this paper except in Section 1.2, when reviewing existing studies in the literature.

1.2. Existing Work and Open Problems

In recent years, the role of scenarios in requirements elicitation, modelling and analysis has been more and more widely recognised [6]. However, scenario-driven requirements analysis has been studied mainly in the context of object-oriented analysis methods. The use of scenarios in requirement analysis originates from Objectory [2,4], and was used in OBJ [2]. The first process model of use case-driven object-oriented analysis was proposed in Jacobson et al. [4]. This model was further developed in Regnell et al. [7] with emphasis on the importance of synthesis of requirements models from scenarios. In Potts et al. [8], the Inquiry Cycle method was proposed to use scenario scripts to identify obstacles or problems in goal-oriented requirements analysis. The SCRAM method proposed in Sutcliffe [9] combined concept demonstrator, scenarios and design rationale. It employed scenario scripts in a walkthrough method that validated design options for ‘key points’ in the script. More recently, based on the concept of Inquiry Cycle and SCRAM, Sutcliffe et al. [10] proposed a method of scenario-based requirement engineering intended to integrate with object-oriented development. The method consists of four stages, which are: (a) elicit and document use case; (b) analyse generic problems and requirements; (c) generate scenarios; and (d) validate system requirements using scenarios. The method relies on the existence of a library of reusable generic requirements attached to models of application classes. A browsing tool matches the use case and input facts acquired from the designer to the appropriate generic application classes and then suggests high-level generic requirements attached to the classes as design rationale ‘trade-offs’. Scenarios are generated by walking through each possible event sequence in the use case and applying heuristics that suggest possible