Theoretical foundations of requirements formalization and verification based on basic protocols are presented. This approach uses the concept of an attributed transition system. Facets of an implementation of the system VRS are described together with some statistical results of using this tool in large-scale industrial projects.

Keywords: formal methods, requirements verification, agents and environments.

INTRODUCTION

In [1], the fixation of requirements is represented as an engineering process determining artefacts that must be produced as a result of a project. This process includes the following stages:

- requirements identification;
- requirements analysis;
- requirements representation;
- requirements communication;
- development of the criteria and procedures of acceptance test of the results obtained.

Requirements are agreements between the customer and designer upon the results that must be obtained during the realization of a project. Requirements must be transparent for the customer and for the designer. The level of their formalization depends on mutual understanding between the parties and on the experience of the specialists involved in identifying the requirements. Unfortunately, the majority of approaches to the description of requirements and preliminary design phase (such as a representation in natural languages, in the form of diagrams, or in a pseudo-code) that are used in practice do not propose software tools for establishing their correctness. Hence, main methods of checking properties of specifications and, accordingly, their behaviors remain inspections and reviews.

In order that a formal analysis be possible, requirements must be formalized, i.e., represented in some formal language. The necessity of formalization imposes definite restrictions on the introduction of formal methods into industry since formalization requires knowledge of logical concepts and notations used in these methods. At present, this knowledge is not widespread among software designers. The use of deductive tools such as PVS [2] is also limited because of the necessity of detailed development of the mathematical theory of the corresponding applied domain for the realization of even sufficiently simple descriptions. To this end, a high level of special mathematical training is required.

An alternative way of using formal methods consists of the representation of requirements not in the form of formal specifications but in design languages, which are widespread in the industry of software development. Formal methods were successfully used for specification in some languages such as Message Sequence Charts (MSCs) [3], SDL [4], and UML [5].

The system PTK [6] automatically analyzes the syntax and semantics of MSCs and, based on these charts, generates test scripts in different languages, including SDL, TTCN, and C. The system FatCat [7] is destined for the detection of a...
nondeterministic behavior in MSCs. In many projects, the initial SDL specifications are previously transformed into some internal representation that makes it possible to use methods of automatic theorem proving and model checking later on. For example, in the system Verimag IF [8], SDL is converted into the language PROMELA, which makes it possible to use the model checking system SPIN. In the company Siemens GSM, protocols are partially verified with the help of the model checking system SVE based on the BDD method [9]. Based on the automatic theorem proving system ACL2 [10], an integrated environment for processing SDL specifications is realized. The companies Telelogic and iLogix that are well known owing to their computer-aided tools of software development participate in the project OMEGA [11] destined for automatic analysis and checking of UML specifications.

Despite appreciable efforts spent for investigations in the field of application of formal methods, the cases when they are inadequate in industrial projects are rather common. This is a consequence of the inefficiency of general decision procedures (supporting the majority of formal methods) in processing bulky specifications.

In this article, an approach based on the concept of basic protocols and the theory of interaction between agents and environments is presented. It is realized in the system VRS (Verification of Requirements Specifications) and was successfully piloted in many industrial software development projects. The VRS system is based on algebraic and insertion programming [12, 13]. The realized system is sufficiently open and allows one to easily adjust it to new object domains. Questions of requirements formalization and verification based on basic protocols are discussed in [14, 15]. The present article supplements the results obtained in [15] with the concepts of concrete and abstract models of basic protocols and relations between them.

Basic protocols represent requirements in the form of Hoare triples \( \alpha \rightarrow < P > \beta \), where \( P \) is a process and \( \alpha \) and \( \beta \) are logical formulas specifying the pre- and postconditions for the process \( P \). The form of Hoare triples is close to the form of requirements and functional specifications used by engineers in practice. The unique difference of Hoare triples from the form of requirements that can be found in technical documentation often is that such requirements use a natural language to express pre- and postconditions and also to describe sequences of actions that form the process of a basic protocol. Elementary requirements formalized in the form of basic protocols are sometimes supplemented with more extensive scripts that describe an expected behavior of the system being designed. Such scripts are usually decomposed into more simple requirements formalized in terms of basic protocols.

The representation of requirements in the form of basic protocols is based on the theory of interaction of agents and environments [16]. In contrast to the majority of traditional interaction theories such as CCS [17], CSP [18], and ACP [19, 20] that are based on an implicit and, hence, unformalized concept of an environment, the theory of interaction of agents and environments studies agents and environments as objects of different types. This theory was first presented in [21]. In this article, we state the theoretical foundations of formalization and verification of requirements with the help of basic protocols and with the use of the concept of an attribute transition system, i.e., a system in which not only transitions but also states are labeled. The basic concepts that are required to work with attribute systems are introduced in Sec. 1. In Sec. 2, the concept of an abstraction of an attribute system is considered. This concept is essential for simulation of large systems.

In Sec. 3, a formal definition of basic protocols and some attendant concepts are introduced. The language of basic protocols is presented in Sec. 4. Concrete realizations are described in Sec. 5. In Sec. 6, an approach to the check of the basic properties of basic protocols, namely, the transition consistency and completeness is stated in general form. In Sec. 7, dynamic properties of systems specified by basic protocols are considered such as the integrity, safety, termination, reachability, and also correctness of annotated scripts. In Sec. 8, some aspects of realization of the VRS system are shortly discussed and statistical results of using its tools in several large-scale industrial projects are presented.

1. ATTRIBUTE TRANSITION SYSTEMS

We consider specifications of interacting systems that are represented in the form of a composition of an environment and agents inserted into the environment. Environments are attribute transition systems. This means that, in addition to a labeling of transitions (by actions), a labeling of states is also used. Formally, an attribute transition system is defined as a collection \( S, A, T, L, l \) of the following components:

- \( S \) is a set of states;
- \( A \) is a set of actions;
- \( T \subseteq S \times A \times S \cup S \times S \) is the set of labeled transitions \( (s \xrightarrow{a} s') \) and unlabeled (hidden) transitions \( (s \rightarrow s') \);
- \( L \) is a set of attribute labelings;
- \( l: S \rightarrow L \) is a partially defined function of labeling states.