A Graph Representation for Use Case Specifications

ONDREJ RYSAVY, FRANTISEK BURES
Faculty of Information Technology
Brno University of Technology,
Bozetechova 2 Brno, Czech Republic

Abstract—Use cases represent widespread industrial approach to formulation and refinement of requirements specification on a system. Although in the last decade several formal representations of use cases were defined there is still gap between their practical use and those theories. This paper provides a formulation of semantics for use cases that is based on the theory of hierarchical graphs arguing that the definition despite its simplicity is powerful enough to represent all common aspects related with use case concepts. Use case specification is divided into three levels. At the bottom level each use case is considered as a flat graph of events. The middle level shapes use cases into episodes enabling to identify and reuse common fragments. The top level depicts interaction between actors and the modeled system through use case entities. Involving hierarchical graphs enables specifying all three levels in the consistent way and provides necessary abstraction on higher levels while all details is maintained in the bottom level.

Keywords: Use cases, Requirements specification, Hierarchical graphs, Formal specification, Formal methods

1. Introduction

The use case concept was recognized as the fundamental tool for capturing the requirements of a system[11] in many popular object-oriented methodologies, such as Objectory[9], SOMA[6], OML[5], UML[13], and OCTOPUS[10]. In spite of the wide consensus on usefulness of use cases in the initial phase of the development, there is not agreement on their standard representation. Use cases can be described informally in natural language, semi-formal in structured natural language, in pseudo code, or with formal models such as activity or sequence diagrams. The most critical point related to the semantics of use cases is investigated in several works which attempt to use formal languages, such as Z language[2], [7], abstract state machines[8], ODAL language [12], or refinement calculus [1], for improving use cases to formal specification technique.

Most of the authors consider use case as a representation of the interaction of the system with external entities, called actors enabling to describe the functionality of the system without revealing its internal structure. In this interpretation use cases model two aspects. The static view considers a system in means of the relations between use cases and actors, and the dynamic view captures behavior as an interaction between actors and the system.

In this paper use cases are considered as a system having tree level architecture [14]. The purpose of the top level is to deal with modeling what actors can use system and what use cases they are using to do so. Conversely, the bottom level reflects dynamic aspects related to interaction between actors and a system. The middle level provides not only a glue putting both level together, but it also structures use cases into fragments called episodes standing for higher-level behavioral units. In this level all common relations usually defined between use case, such as uses, extends, and invokes relations, can be applied to episodes. This separation of views enables to cope with different kind of information (static structure, services, behavior) but to keep the specification consistent.

The paper is organized as follows. The next section includes an overview on hierarchical graphs as a preparation for section 4 in which basic specification of use case concepts is presented and briefly demonstrated. The paper ends with some concluding remarks.

2. Hierarchical graphs

Hierarchical graphs [3],[4] extend capabilities of ordinary graphs enabling representation of structural information such as grouping and encapsulation, which cannot be easily modeled in the ordinary flat graphs. In this section an overview of the hierarchical graphs is given in order to establish definitions for the rest of the paper.

Let \( \Sigma \) be a fixed set, called the node alphabet. A directed graph with labeled nodes is a tuple \( G = (N, E, s, t, l) \), where \( N \) is a set of nodes, \( E \) is a set of edges, \( s : E \rightarrow N \) and \( t : E \rightarrow N \) are two functions mapping an edge to its source node and its target node respectively, and \( l : N \rightarrow \Sigma \) is the node labeling function. A loop in the graph is an edge \( e \) with \( s(e) = t(e) \). An unlabeled directed graph is a tuple \( G = (N, E, s, t) \) where node-labelling function was dropped. An unlabeled directed graph is called simple graph if it contains no loops and multiple edges. Such graph can be defined simply as \( G = (N, E) \), where \( N \) is a set of nodes as defined above and \( E \subseteq N \times N \) is a set of edges.

A path in \( G \) from \( u \in N \) to \( v \in N \) is a sequence of edges \( e_1, \ldots, e_k \) for some \( k \geq 1 \) such that, for all \( i = 1, \ldots, k-1 \), \( t(e_i) = s(e_{i+1}) \), \( s(e_1) = u \), and \( t(e_k) = v \). A cycle in \( G \) is a path from a node \( n \in N \) to itself.

A directed acyclic graph is a graph \( G = (N, E, I) \) that contains no cycles. Given a directed graph \( G = (N, E, I) \), if there exists a node \( n \in N \) such that, for all \( m \in N \), there exists a path from \( n \) to \( m \) in \( G \), then we say that \( n \) is a root of \( G \) and we call \( G \) a rooted directed graph.

A simple graph \( B = (N, E) \) is called bipartite graph, if set \( N \) can be partitioned into two sets of nodes \( X, Y \), such that,
for all edges $e = (x, y) \in E$, either $x \in X$ and $y \in Y$ or $x \in Y$ and $y \in X$.

A simple directed bipartite graph $C = (X_C, Y_C, E_C)$ is a coupling graph if it satisfies completeness condition asserting that for every $x \in X_C$ there exists some $y \in Y_C$ and an edge $e = (x, y) \in E_C$, and injectivity condition claiming that for every $x \in X_C$, if there exists some $y \in Y_C$ and an edge $e = (x, y) \in E_C$, then for all $e' = (x', y') \in E_C$ $x' \neq x$ and $y' \neq y$, or $e = e'$.

The completeness condition assumes that every node of set $Y_C$ is connected to at least one node in $X_C$. The injectivity condition requires that every node in $X_C$ is connected to at most one node in $Y_C$ but no two nodes in $X_C$ can be both connected the same node in $Y_C$.

A hierarchical graph is a triple $H = (G, D, C)$, where $G$ and $D$ are graphs with sets of nodes $N_G$ and $N_D$ respectively, and $C = (X_C, Y_C, E_C)$ is a coupling graph such that $X_C = N_G$ and $Y_C = N_D$. The graph $D$ is called the hierarchy graph, while $C$ is coupling graph of $H$.

3. A Use Case Specification

In this section a theory for use case models is described using definition of graphs from the previous section. In [14] two complementary approaches to creation of use case models are described together with proposing three level architecture, which corresponds closely to the theory presented here. The architecture divides an use case specifications into the three levels, each representing different view. Before the architecture for use cases can be formally defined, all related concepts considered as modeling elements need to be enumerated for clarifying conceptual framework for the developed theory. They can be classified according to their intended role that will play in the models. As the result of the classification a collection of classes each standing for the particular sort of modeling entities and their mappings is obtained. Informally, classes represent concepts of use case specifications with the meaning as following:

- **Actor** participates actively on behavior of the system. Each actor is associated with one or many use cases which denotes an usage relation.
- **Use case** is complementary modeling element to actors. It defines an isolated sequence of action that can be performed on the system in order to achieve some goal.
- **Episode** is a named use case part. An episode can be expanded into new episode structures. Once episode structure is defined it may be shared among other episodes. Episodes relations describe episodes from their structural viewpoint. The extends relation expresses that one episode is extension of another, the uses relation indicates that an episode is contained in the other episode, and invokes relation specifies an order in which related episodes are executed.
- **Event** represents an occurrence of communication between an actor and the system or isolated action in the system. Events participate only in the lowest level of the use case architecture. The possible behavior of the system within use case is unambiguously defined by the event occurrences and control flow.

- **Stimulus** specifies sending a message from an actor to the system. Messages may carry additional information as arguments or may only initiate series of actions.
- **Response** specifies sending a message from the system to an actor. As in the case of stimuli responses may carry additional information.
- **Action** provides information on the internal behavior of system. Although use case specification often consider system as a black box, which internal behavior is hidden to user, definition of actions enables to specify significant behavior of the system related to performed task.

Once elements of use case schema are classified, their formal representations is refined in the notation of set theory. It prepares formal framework for defining abstract use case specification. The following definition constructs such framework by declaring classes of entities as sets and the collection of related operators as functions on these sets.

Let $A$ be a set of actors, $U$ be a set of use cases, $E$ be a set of episodes, $X$ be $S \cup R \cup T$ be a set of events, where $S$ is a set of stimuli, $R$ is a set of responses and $T$ is a set of actions, and $\Delta$ be a set of strings (or more precise string identification), then $\eta : U \cup E \rightarrow \Delta$ is a labeling function assigning to each use case or episode a name.

The architecture of use case specification consists of three layers each providing a different view on a functionality and structure of the abstract system. Again all layers of the architecture and its purpose are informally explained first. Each layer should be understood as a particular view on the system properties rather than the definition of a system architecture.

- **User level** is the topmost layer of the use case architecture. It collects all actors participating in the behavior of a system, and use cases specifying system’s behavior as activities. The usage relation joins an use case with the actor, which is the initiator of the activity specified by the use case.
- **Structure level** describes internal structure of each use case as a hierarchy of episodes. It uses episode relations to provide abstract structure information on use case arrangement with possibility for reuse and share of episodes.
- **Event level** uses control flow graph of events to define activity of the whole system. Event level does not specify structure information nor any boundaries which are left to be provide by higher levels. Event level consists of event entities and defines operators for mapping stimuli and responses to corresponding actors.

Obviously, the theory of hierarchical graphs can provide a natural representation for just described use case architecture. The graph representation of the use case specification involves three flat graphs, one for each level, and two coupling graphs gluing adjacent levels.
Fig. 1. Symbols used in the use case architecture graph

The top architecture level requires to distinguish two sorts of nodes - the actor nodes and use case nodes. Simple undirected bipartite graph seems to fit well to this purpose. Since each use case is associated with exactly one actor the incidence relation is restricted by additional constraint.

A bipartite undirected graph \( G_2 = (U, A, I) \) represents a user level graph if two sets of nodes \( U, A \) are sets of use cases and actors and \( I \subseteq U \times A \) is the incidence relation satisfying that for all \( (u, a) \in I \) and \( (u', a') \in I \), if \( u = u' \) then \( a = a' \).

The middle level constitutes layer in which use cases are structured into episodes. It has form of directed graph with labeled edges. Labeling supplies a kind of relation represented by an edge. The multiple edges are not generally forbidden although it is not possible to associated two episodes by arbitrary combination of relations simultaneously.

An edge-labeled directed graph \( G_1 = (E, \Lambda_1, s_1, t_1, p) \) represents a structure level graph, if \( E \) is a set of episodes, \( \Lambda_1 \) is a set of relations on episodes which is disjunct from any other defined sets, \( s_1 : \Lambda_1 \to E \) and \( t_1 : \Lambda_1 \to E \) are incidence functions, and \( p : \Lambda_1 \to F \) is the edge-labeling function mapping to each edge its kind from the set of kinds \( F \) of episode relations.

The lowest level of the use case architecture forms an event graph that depicts control flow, which is modeled as graph’s edges, and occurrences of events, which are associated with nodes. Node mapping function \( l \) enables multiple occurrence of an event in the graph, which would not be possible if nodes stood directly for events.

A node-labeled directed graph \( G_0 = (\Gamma, \Lambda_0, s_0, t_0, l) \) represents an event level graph, where \( \Gamma \) is a set of events, \( \Lambda_0 \) is a set of edges indicating control flow, \( s_0 : \Lambda_0 \to \Gamma \) and \( t_0 : \Lambda_0 \to \Gamma \) are incidence functions, and \( l : \Gamma \to X \) is the node-labeling function assigning to each node of the graph an event from the set \( X \).

An example from figure 2 depicts one use case specification. It is part of the authorization system used by employees to gain admission to a building. The user level represented by the graph \( G_2 \) consists only of the employee node, the open door use case, and the usage relation denotes that employee uses open door use case. The structure level bounded by graph \( G_1 \) divides actions related to open door use case into two episodes that are ordered by the invoke relations. The lowest level \( G_0 \) describes detailed possible actions and communications of the whole use case. Labeling is denoted by the horizontal dashed line, while inter-layer group association is given as a dot-dashed vertical line. It can be seen that nodes \( e_1 \) and \( e_2 \) are grouped in the draw card episode while the nodes \( e_3, e_5, e_6, e_7 \) belongs to enter code episode.

4. Conclusion

In the paper a formalism for use cases based on hierarchical graphs was given. It establishes three different conceptual levels for specification of use cases that cover both static and dynamic views on requirements of a system. The top level expressing relations between users of a system denoted as actors and services of a system represented as use cases is modeled as bipartite graph. The bottom level capturing
behavior of a system is defined as directed graph whose nodes are labeled with events. The middle level describes an abstract behavior by grouping nodes of underlaying graph. It has a form of directed graph in which edges are labeled by one kind of the episode relations.

A contribution of the presented work is providing abstract formal representation for use case concepts that do not restrict further development, i.e. new concepts may be added if they can be classified into one of the three level or they even may specify another inter-level relations. Graph representation is natural for use case concepts giving formal semantics to visual information, which represents current trend in many modeling methodologies. The paper does not address possible representations for related modeling concepts such as static attributes of actors, messages or actions, which is out of the scope of the described abstract use case specification architecture.

5. Acknowledgements

This research has been partly funded by the Czech Ministry of Education in frame of the Research intention No. CEZ: J22/98: 262200012 - Research in information and control systems and by the grant FRVS 838/2003: Tools for supporting formal specification and verification in UML diagrams by the Czech Ministry of Education.

References: