Transformation and Integration Method of Scenarios

Atsushi Ohnishi
Department of Computer Science, Ritsumeikan University
Kusatsu, Shiga 525-8577, Japan
Tel:+81-77-561-2693, Fax:+81-77-561-2669, ohnishi@cs.ritsumei.ac.jp

Abstract  Scenarios that describe concrete situations of software operation play an important role in software development, and in particular in requirements engineering. Scenario details should vary in content and detail when described from different viewpoints (e.g. types of user or external interface), but this presents a difficulty, because informal scenarios cannot easily be translated from one viewpoint to another with consistency and assurance. This paper describes (1) a language for describing scenarios in which simple action traces are embellished to include typed frames based on a simple case grammar of actions, and (2) a procedure for translating a frame-based scenario from one viewpoint into another, and (3) integration method of scenarios described from different viewpoints. We illustrate them with examples of program chair’s job.

Key Words: scenario, requirements definition, software requirements specification, and viewpoint

1. Introduction
Scenarios have important uses in software specification and design synthesis, particularly when used to gain insight into the implications of design decisions and for customer validation and acceptance. However, scenarios are vague and imprecise tools and are therefore difficult to integrate into more formal development processes.

For example, without defining what it means for actions to be included in a scenario, it is impossible to compute differences among scenarios reliably. (Is a difference because of an unimportant omission, or a change in names, or a differently instantiated variable, or is it a genuine inconsistency.) Nor is it possible for the scenarios to guide the refinement of a specification other than by providing insight to the specifier. Detailed, instantiated scenarios of the kind used for customer validation of requirements or the generation of system test cases can’t be generated mechanically from scenarios described just in text. Most importantly vulnerabilities of the spec/scenario system are not easily flagged. (E.g. what kinds of things might go wrong in the performance of this action? What previously executed actions does this scenario or scenario
We claim that scenarios should be represented in a form that is closer to the application-specific concepts relevant to customer validation and system testing than general-purpose concepts such as "actor", "step", "event", "episode", "obstacle", and etc. But going to the extreme of application-specific representations relies on domain-specialists investing work in the development of application-specific (or still worse, system-specific) languages and tools. An intermediate, semantically rich, but application-neutral set of concepts is what's needed.

Extensions to scenario and action representations have been proposed before. Ohnishi’s requirements frame model[9] provides a vocabulary of general-purpose action and information types using structures similar to those of case grammar[4].

The requirements frame model defines the case structure of a concept. For example, the data flow (DFLOW) concept has agent, source, goal, and instrument cases. The agent case corresponds to data that are transferred from the source case object to the goal case object. So, an object assigned to the agent case should be a data type object. An object in the source or goal cases should be either a human or a function type object. If and only if a human type object is assigned to source or goal cases, some instrument should be specified as a device case. The requirements frame enables to detect illegal usages of data and lack of cases[9].

In this paper, we outline a frame-based approach for structuring the actions in a scenario. We use ideas from previous work[9] and Jackson's problem frames[6] for structuring the content of scenarios and action descriptions in specifications. Thus, there are different case frames for different problem frames, but far fewer case frames than there are systems or problem domains.

In this respect our "middle-level" approach to describing system behavior resumes Halliday's approach[5] to the transitivity structure of natural language by rejecting attempts to model all behavior according to a single set of primitive action types while avoiding the profusion of action types that would result from an application-specific language.

We describe our approach in the following two sections, concentrating in turn on the types of entity and role that appear in scenarios as actors or passive objects and then on the types and patterns of actions through which these entities/roles interact. In each of these two sections, we describe the entities/roles and action types relevant to the distinct problem frames we identify. We then present a familiar example: the jobs of international conference program chair. This problem is also adopted as a common example of the Working Group of Requirements Engineering, IPS, Japan.

2. Scenario description language

We have developed a scenario description language based on the case grammar. We specify a sequence of events as a scenario with this language. Each of events has just one verb, and each of
verbs has its own case structure. Verbs and their own case structures are depend on problem domains, but roles of cases are independent of problem domains. There exist several roles, such as agent, object, recipient, instrument, source, and so on.

An entity doesn't have one of these as a type, but plays these as roles. (Thus a person may be a user w.r.t. requesting services of the system, but a consumer of its outputs.) There are some constraints. Thus, we expect entities either to be users or phenomena, but not both w.r.t different actions.

We provide action frames as shown in Table 1.

<table>
<thead>
<tr>
<th>action cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform/derive/compute agent, source, goal, object</td>
</tr>
<tr>
<td>Collate/compare                  agent, source, object</td>
</tr>
<tr>
<td>Report on/monitor/sense          agent, source, operation, instrument</td>
</tr>
<tr>
<td>Query/ask                       agent, object, goal</td>
</tr>
<tr>
<td>Command/control                 agent, recipient, operation, instrument</td>
</tr>
<tr>
<td>Suggest/request                  agent, object, goal</td>
</tr>
<tr>
<td>Create/edit                     agent, source, goal</td>
</tr>
<tr>
<td>Feedback                        agent, recipient, operation, instrument</td>
</tr>
<tr>
<td>Inform/notify                   agent, object, goal, instrument</td>
</tr>
<tr>
<td>Allocate/schedule/seize         agent, object, goal</td>
</tr>
<tr>
<td>Cancel/relinquish                agent, recipient, operation, instrument</td>
</tr>
<tr>
<td>Exist/locate                    agent, status</td>
</tr>
<tr>
<td>Move/transfer/input/output      agent, source, goal, instrument</td>
</tr>
<tr>
<td>React                           agent, operation, instrument</td>
</tr>
</tbody>
</table>

Just like Ohnishi's Case Frame[9], each of actions has its case structure. For example, action "move" has four cases, such that "agent," "source," "goal," and "instrument." A sentence "Mr. X moves from Tokyo to London by airplane" can be transformed into an internal representation as shown in Table 2. We assume that a scenario represents a sequence of events, and each of events can be transformed into internal representation based on the above action frame.

<table>
<thead>
<tr>
<th>Action</th>
<th>agent</th>
<th>source</th>
<th>goal</th>
<th>instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>move</td>
<td>Mr. X</td>
<td>Tokyo</td>
<td>London</td>
<td>airplane</td>
</tr>
</tbody>
</table>

3. Scenario example: program chair's job

In this chapter, we consider a scenario of program chair's job at an international conference. This scenario is based on a problem provided by Requirements Engineering Working Group (RE WG), Information Processing Society, Japan. Similar job description is provided by ACM[1]. First we show an informal scenario description, and then show a formal representation based on the action
3.1 job description

The followings are program chair's jobs.

1) He decides a schedule of the paper submission deadline, program committee's meeting, acceptance/rejection notification, camera-ready paper due, and sends the schedule to a publishing company.

2) He arranges keynote speakers.

3) He makes Call for Paper, and distributes it to related societies and organizations.

4) He recruits and appoints program committee members, and makes a list of members including member's name, address, affiliation, phone number, FAX number, e-mail address, and research area.

5) He assigns paper IDs to submitted papers. Each of submitted papers has title, author name, address, affiliation, phone number, e-mail address, FAX number, abstract, and key words.

6) He sends acceptance mails to authors via e-mail or postal mail.

7) He assigns reviewers after the submission deadline. The strategy of the assignment is as follows.
   (a) He sends a list of papers to all the PC members.
   (b) PC members select and notify their candidates of papers for reviews.
   (c) PC chair makes a final assignment of reviewers. Each paper should be reviewed by three PC members.

8) He sends reviewer's sheets and submitted papers for review to PC members.

9) He receives reviewer's results

10) ...

3.2 scenario from PC chair's viewpoint

From this job description we can derive a scenario from PC chair's viewpoint by selecting events whose subject is PC chair in the description. In this derivation data structures, such as list of PC members are omitted. Events whose subject is not PC chairs, such as “PC members select their candidates of papers” are omitted.

   c1) He decides a schedule.
   c2) He sends the schedule to a publishing company.
   c3) He arranges keynote speakers.
   c4) He makes Call for Paper, and distributes it to related societies and organizations.
   c5) He recruits and appoints program committee members, and makes a list of members.
   c6) He assigns paper IDs to submitted papers.
c7) He sends a list of papers to all the PC members.
c8) PC members notify papers for reviews.
c9) He assigns reviewers.
c10) He sends reviewer’s sheets and submitted papers for review to PC members.
c11) He receives reviewer’s results

3.3 scenario from PC member’s viewpoint

The followings are jobs of PC members.
m1) PC member is selected by a PC chair and receives a request from the PC chair.
m2) The PC member sends an acceptance as a PC member and also replies information about his name, postal address, e-mail address, and so on to the PC chair.
m3) The PC member receives a list of submitted papers.
m4) He notifies a list of papers which he wants to review.
m5) The PC chair decides reviewers for each of the submitted papers.
m6) PC members receive papers to review and reviewer’s form.
m7) PC members review the papers.
m8) They send the reviewing result of papers.
m9) ...

4. Transformation and integration of scenario

4.1 Transformation of events in scenario

In the previous scenarios, some events can be represented from PC chair's viewpoint and from PC members' viewpoint. For example, (1) PC chair sends a list of submitted papers to PC members (from PC chair’s viewpoint) is the same as (2) PC members receives a list of submitted papers from PC chair (from PC members’ viewpoint.) These events correspond to the same data flow. In case of data flow, there exist source case object, goal case object, and agent case object. So, each of data flow events can be easily transformed and represented from different viewpoints, such as source case object's view and goal case object's view. For example, the above event can be transformed into as below.

<table>
<thead>
<tr>
<th>Action</th>
<th>agent</th>
<th>source</th>
<th>goal</th>
<th>instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>transfer</td>
<td>List of papers</td>
<td>PC chair</td>
<td>PC members</td>
<td>-</td>
</tr>
</tbody>
</table>

Just like natural language processing, we can transform events in scenarios from a certain viewpoint into events from another viewpoint using the action frame. In case of data flow events,
there exists a simple transformation rule, namely to say,

The event “Source case object send inactive agent case object to goal case object with instrument case“ becomes “Goal case object receive inactive agent case object from source case object with instrument case” and vice versa. With this rule, it is quite easy to transform between an event from source case object’s viewpoint and an event from goal case object’s viewpoint. However, in case of another problem domain, we have to establish different transformation rules. In [11], the elevator control problem[16] is described from several points of view, including the human passengers, mechanisms (sensors, motors, etc.), inanimate places and vehicles (the floors and elevator cars) or anthroporphically from the controller's point of view. We apply the frame-based approach to several complementary and overlapping views of elevator use scenarios, showing how errors and omissions in one scenario may be corrected by walking through the same scenario from other viewpoints. In this case, we found four transformation rules[11].

4.2 Scenario integration

Since events with the action frame are independent of a certain viewpoint, we can integrate and generate a new scenario as shown in Fig.1.

![Fig.1 Scenario integration.](image)

We can integrate a new scenario from PC chair’s viewpoint from the above two scenarios as below.

1) **PC chair decides a schedule.**
2) **He sends the schedule to a publishing company.**
3) **He arranges keynote speakers.**
4) **He makes Call for Paper, and distributes it to related societies and organizations.**
5) **m1) He recruits and appoints program committee members, and makes a list of members.**
6) **m2) He receives acceptance letters including information from PC members.**
c6) He assigns paper IDs to submitted papers.
c7, m3) He sends a list of papers to all the PC members.
c8, m4) PC members notify their candidates of papers for reviews.
c9, m5) He assigns reviewers.
c10, m6) He sends reviewer's sheets and submitted papers for review to PC members.
c11, m8) He receives reviewer's results

c12) ...

The event labeled “m2” was missing in the scenario in section 3.2. We can point out the lack of events by integrating scenarios from different viewpoints.

5. Conclusions

We have developed an action frame base scenario description language, scenario transformation method, and integration method of scenarios from different viewpoints. With this method, we can get a new scenario from PC chair’s viewpoint by transforming a scenario from PC member’s viewpoint and integrating a scenario from PC members’ viewpoint and a scenario from PC chair’s viewpoint. Through our example, we found the scenario transformation and integration with action frame enables to improve several quality of scenario.

1) System developer does not need to describe scenarios from several different viewpoints. If he specifies a scenario from one viewpoint, then he can get a scenario from another viewpoint with transformation rules. This contributes to improve the productivity of scenarios.

2) If system developer specifies two scenarios from different viewpoints, he can check the lack of events and check the consistency of scenarios by integrating scenarios from different viewpoints. This contributes to improve the completeness and the consistency of scenarios.

We have to develop scenario description language, description language for transformation rules and their processors. We have to validate the ideas more thoroughly by applying to several different problem domains.

Acknowledgement

The author heartily thanks to Associate Professor Colin Potts at College of Computing, Georgia Institute of Technology, Atlanta, Georgia, U.S.A. for his valuable comments. He also thanks to his graduate students of Ritsumeikan University, Mr. Zhang Honghui, and Mr. Hiroshi Fujimoto for their contributions to this research.
References


[4] Fillmore, C.J.: &quot;The Case for Case,&quot; in Bach and Harms (Eds.), &quot;Universals in Linguistic Theory,&quot;


