Decoupled Model-Based Elicitation of Stakeholder Scenarios

Gregor Gabrysiak, Regina Hebig, and Holger Giese
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{gregor.gabrysiak|regina.hebig|holger.giese}@hpi.uni-potsdam.de

***Abstract*— Requirements engineers iteratively elicit scenarios by capturing and combining individual stakeholder perspectives into a consistent overall scenario model. This model has to be validated to exclude elicitation errors and check whether all alternatives are covered. While involving all stakeholders at once is considered beneficial, it is usually not feasible due to scheduling and resource constraints. Consequently, techniques that permit all stakeholders to be involved in the elicitation and validation independently, i.e., temporally and locally decoupled, are required. In this paper, we present an approach that enables stakeholders to participate in the elicitation of their collaborative scenarios remotely and decoupled from other stakeholders. The resulting fragmentation of the elicited scenarios is overcome by allowing stakeholders to express their expectation on how a scenario is usually complemented by activities of other stakeholders. Our approach systematically combines these decoupled perspectives to establish the overall scenario model.**

Keywords-decoupled requirements elicitation; scenario synthesis; incomplete scenarios.

I. Introduction

A requirements engineer *gets people to tell the stories of what their systems are meant to do*, as pointed out by Alexander and Maiden [2]. For complex systems with multiple, collaborating stakeholder groups, a requirements engineer needs to listen to all of their stories and to synthesize these stories into suitable scenario models. Among other aspects, these scenarios have to capture how the involved stakeholders interact to achieve their common goals.

The requirements engineers start by eliciting scenarios from individual stakeholder perspectives. After combining these separate scenarios into a consistent overall scenario model, they validate this model to exclude elicitation errors and check whether they covered all alternatives. Then, these initial activities continue iteratively with additional elicitation activities, updates of the scenario model, and subsequent validation activities until the result stabilizes. The overall scenario model is the crucial element to ensure that a consistent understanding of the different stakeholder perspectives can be established.

The elicitation and validation of such scenarios requires less effort if all stakeholders are involved simultaneously. By directly commenting on whether they agree with the statements of other stakeholders, the requirements engineer might obtain a commonly agreed-upon scenario model directly within an elicitation session [15]. However, due to scheduling and resource constraints such a setting is usually not feasible, if not less efficient compared to elicitations with individual stakeholders [17]. Also, experience shows that in case of group meetings social effects can result in suppressing opinions and observations of stakeholders positioned lower in the hierarchy. Furthermore, the stakeholders who participate are sometimes chosen based on who is noncritical for the daily work to continue without interruptions [1]. To limit such effects, techniques that permit all stakeholders to be involved in the elicitation, consistency, and validation without the necessity to be present in person at the same location and at the same time are required.

In our former work we developed a model-based approach [9]. After initial elicitation interviews, an overall scenario model is set up and can be validated by stakeholders in an interactive simulation (*play-out*, cf. [12]). Stakeholders can complement each other's activities through refinement or playing in additional activities, which result in consistent model updates. Still, the initial elicitation of new scenarios remained a problem. Since the simulator cannot know how to react if no suitable response was observed before, stakeholders run into dead ends during their elicitation sessions (referred to as *stalemates*). Arrange a meeting of all involved stakeholders to elicit the new scenarios in one simulation session is a complex, time-consuming solution. If this is not feasible, the only alternative is to perform multiple simulation sessions with individual stakeholders. To play-in her parts of one scenario, a stakeholder is enforced to participate in multiple sessions, waiting in-between for other stakeholders to play-in their continuations for the scenario. Especially for stakeholders playing coordinating roles, this can lead to numerous sessions.

In this paper, we present an extension of the simulation approach, that overcomes this challenge and can reduce the number of necessary sessions. Therefore, we use the simulator's property to not capture scenarios in an explicit process view, but in form of reachable states and possible transitions between them. The idea is to empower stakeholders to explicitly express the responses they usually expect when interacting with other stakeholders in form of partial states. Based on a partial state, a stakeholder can continue playing-in her parts of the scenario, without requiring additional elicitation sessions. The extension of the simulator is able to recognize the fulfillment of such expectations, thus, iden-

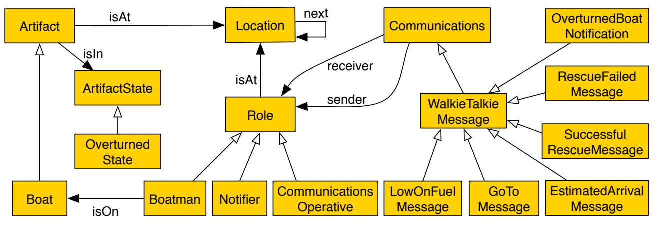


Figure 1. (Abridged) ontology $\mathcal{D}_W$ elicited from a lifeguard service

tifying a suitable continuation of incomplete interactions. Consequently, the results of multiple simulation sessions can be combined automatically.

The paper is structured as follows: At first, we present our model-based approach for the validation, additional elicitation and model update of scenario models and discuss its limits in Section II. Based on practical examples elicited from a lifeguard service, we discuss our approach on how stakeholders can describe their *expected continuations* by simply answering three questions, thereby overcoming these limitations in Section III. Then, two different types of *triggers* are presented in Section IV. Section V explains how such triggers are fulfilled and how the overall scenarios can be synthesized automatically. The paper closes with a discussion of related work in Section VI and a conclusion.

II. Model-Based Requirements Validation

Requirements engineers can hope to solve the stakeholders' problems only if they capture the concepts of the stakeholders' domain correctly [4]. This can be achieved by a domain ontology $\mathcal{D}$, which is gathered by the requirements engineers similar to a glossary of commonly agreed upon definitions of concepts. By collecting all concepts of the domain under investigation, the requirements engineers obtain a model suitable to describe scenarios in that domain. Similar to Artifact Models [3], $\mathcal{D}$ also captures how these concepts relate to each other. As outlined in [7], a domain ontology $\mathcal{D}$ contains the roles involved and identified from their scenarios, the artifacts that they use, and the specific information that they share. For our example, Figure 1 illustrates the abridged version $\mathcal{D}_W$ of the domain ontology elicited from a lifeguard service. In this example, the communication between a bystander notifying the lifeguards (referred to as *Notifier*) about an emergency, the corresponding communications operative (*ComOp*), and a boatman are elicited.

Kühne [14] argues, that metamodels such as $\mathcal{D}$ specify a language that can be used to describe instance situations. Thus, $\mathcal{D}_W$ provides a language for describing states as they can be observed during the scenarios of the lifeguards.

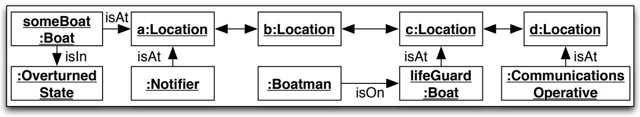
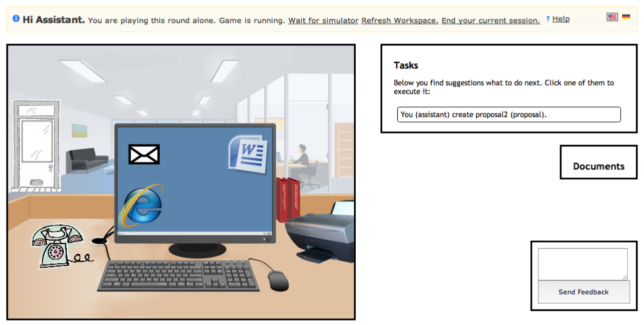


Figure 2. An initial state s_{init} of the lifeguard service scenarios


Hi Assistant. You are playing this round alone. Game is running. Wait for simulator Refresh Workspace. End your current session. ? Help

Tasks

Below you find suggestions what to do next. Click one of them to execute it:

You (assistant) create proposal2 (proposal).

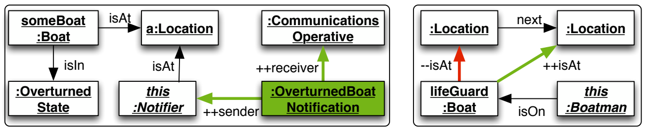
Figure 3. An example of a workplace visualization that allows stakeholders to interact with other, simulated roles to validate their interactions [6]

Similar to a Unified Modeling Language (UML) Class Diagram, $\mathcal{D}_W$ prescribes all possible states of the scenarios. States, in turn, can be specified using UML Object Diagrams. Such a state is illustrated in Figure 2. It is also the initial state s_{init} referred to in the sequence diagrams throughout the paper. In the following, all state labels in sequence diagrams refer to complete or partial states represented by such object diagrams.

Based on the idea of Harel and Marelly's *play-out* [12], our approach includes a simulator, which is able to play-out behavioral specifications to simulate the behavior observed beforehand from specific stakeholders. This simulator allows participating stakeholders to experience interactions with other stakeholders who are not participating in the same simulation session. Our simulator [9] decouples these interactions temporally by replaying them using the specifications to complement activities of participating stakeholders and, thus, allows stakeholders to validate each other's behavior without having to be in the same session or the same room.

Each stakeholder participating in our simulation has an individual interactive visualization (Figure 3, cf. [6]) of their distinct perspective on the current state of the simulation. Depending on the considered domain, different concepts are visible at different points in time. In case of the lifeguards, boats can only be seen if the stakeholder is at the same location (cf. Figure 1). The same holds for different artifacts or even other stakeholders. Thus, what has to be visualized to reproduce a distinct stakeholder's individual perspective is domain-specific. However, the requirements engineers can prototype what has to be shown to quickly get the details right. Then, the same rules usually apply for all stakeholders.

By using the visualization to interact with the simulation, participating stakeholders can change the state of the simulation according to what they would normally do in a corresponding situation, e.g., a boatman might move on to the next location or upturn an overturned boat. This visualization allows stakeholders to play through scenarios by interacting either directly with their colleagues or with roles that are simulated based on prior observations. In a related experiment, the state visualization has been evaluated successfully [8]. Thus, the play-out enables stakeholders to validate what has been observed and captured so far.



(a) Notifying the life guard service (b) Going to the next location

Figure 4. Story patterns of a $Notifier$ informing a CommunicationsOperative about an overturned boat he sees *(a)* and of a Boatman going from one location to the next *(b)*

After stakeholders have played through one of the valid scenarios, all states observed in-between their initial state s_{init} and the final state describe the activities of individual stakeholders and how they interacted. To formally capture what happened between succeeding states, our approach relies on graph transformations (specifically *story patterns*). Graph transformations, as described by Heckel [13], consist of a precondition and a postcondition. Similar to an object diagram, the precondition is a structural specification. If a state contains an exact match for the precondition of a story pattern, the matching elements are restructured through the addition or deletion of the corresponding elements and associations to match the story pattern's postcondition, which is also a structural specification. In story patterns, the elements that need to change are color-coded. Additions, i.e., instances and associations which have to be added, are marked using green and "++", while "– –" in red indicates removals. For example, Figure 4 illustrates a Notifier notifying a ComOp (*a*) and a Boatman changing locations (*b*).

By observing scenarios, i.e., sequences of states, the simulator can automatically derive story patterns based on the changes between two succeeding states. Each story pattern is then assigned to a specific role, i.e., to the one represented by the stakeholder, who was observed executing the behavior that changed the state of the simulation. In each story pattern, the instance of the specific role it belongs to is named *this* (cf. Figure 4).

As mentioned before, each stakeholder participating in a simulation has a unique perspective on a state s_i during a simulation session. After each activity of another role, the stakeholder might be affected by the result. Still, only some of these activities and their results are even visible to the role that this stakeholder represents. Consequently, every time a stakeholder participating in the simulation is affected by a change of the current state of the simulation, e.g., when a boat arrives at his location or an artifact is brought to his attention, this change has to be reflected in the stakeholder's visualization. This visualization illustrates the current state s_i of the simulation reduced to what a stakeholder's role is able to perceive. We refer to the reduction of a state s_i to what is visible for a role $Role_T$ as *projection*. Thus, a partial state $s_i|_{Role_T}$ can be derived from a state s_i using the visibility information, which apply for a domain while $s_i|_{Role_T} \subseteq s_i$ has to hold. Per default, $s_i|_{Role_T}$ contains an instance of $Role_T$ itself as well as all artifacts and information this role has access to in s_i.

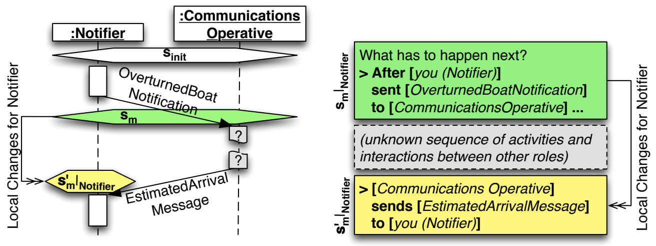

:Notifier
:Communications Operative
s_{init}
OverturnedBoat Notification
s_m
?
?
$s'_m|_{Notifier}$
EstimatedArrival Message
Local Changes for Notifier
$s_m|_{Notifier}$
What has to happen next?
> After [you (Notifier)]
sent [OverturnedBoatNotification]
to [CommunicationsOperative] ...
(unknown sequence of activities and interactions between other roles)
$s'_m|_{Notifier}$
> [Communications Operative]
sends [EstimatedArrivalMessage]
to [you (Notifier)]
Local Changes for Notifier

Figure 5. While the $Notifier$ knows how she informs the $ComOp$ (*green*), she does not know how the $ComOp$ continues this scenario (*gray*); still, the $Notifier$ can validate his expectations, i.e., how he is affected (*yellow*), in a natural language representation (*right*)

To allow stakeholders to comment on story patterns or describe what they can perceive or expect directly, the underlying object diagrams offer a *Natural Language* representation that is easier to understand for stakeholders as illustrated in Figure 5.

Based on stakeholder observations of *what* they do and *how* they do it, the story patterns derived from these observations can be used to replay and simulate the behavior observed from the individual stakeholders. This, in turn, enables the simulator to employ strategies to simulate other stakeholders and, thus, to steer the simulation into conflicting or unresolved states. Consequently, stakeholders can validate the behavior of other stakeholders by either agreeing to it or pointing out errors. Through the simulation of other roles, it becomes unnecessary to get all interacting stakeholders together in one room at the same time. Attending the simulation does not even require the attendees to be at the same location, since the visualization is web-based and, thus, allows for remote sessions [6]. Since the stakeholders can play-in incomplete scenarios, which can then be used to simulate them during simulation sessions with other stakeholders, the simulator also decouples the stakeholders temporally. By providing a model-based validation for behavioral models describing collaborative scenarios, our simulator tries to solve the problem of elicitation and validation for complex systems with multiple collaborating stakeholders.

If a stakeholder is observed starting an alternative scenario the simulator cannot respond appropriately, i.e., cannot offer any reasonable continuations. Since no behavior is available that completes this unknown situation, the requirements engineers have to talk to other stakeholders first to get to know a reasonable continuation for this scenario. Still, the remainder of the scenario might be unknown as well. Consequently, in the worst case, the requirements engineers have to go back and forth between different stakeholders to complete this scenario. In the worst case, to elicit a simple scenario between two stakeholders, each of them might have to be interviewed once for each interaction they have.

III. Approach: Changes in Partial States

As illustrated in Figure 6, the requirements engineers have to deal with individual perspectives as well as handovers. It

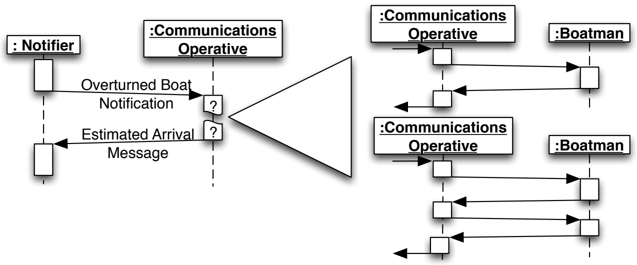

: Notifier
:Communications Operative
Overturned Boat Notification
?
Estimated Arrival Message
?
:Communications Operative
:Boatman
:Communications Operative
:Boatman

Figure 6. The scenario on the left (a *Notifier* calling in an emergency) is incomplete and the Notifier cannot know, which of the possible continuations (*right*) will occur

TABLE I
BY ANSWERING THESE QUESTIONS, STAKEHOLDERS DESCRIBE, WHICH INTERACTIONS USUALLY OCCUR AND HOW THEY END

	Question for Stakeholder	*Answer of Notifier*	*Named*
Q_1	Who did you interact with?	CommunicationsOperative	$Role_{Req}$
Q_2	Who, if not [$Role_{Req}$], do you expect to get an answer from?	CommunicationsOperative *(default: $Role_{Req}$)*	$Role_{Resp}$
Q_3	How are you affected by the outcome? What do you expect [to get]?	"An Estimated Arrival Message is sent to me"	$s'_m\|_{Role_T}$

happens quite often, that a stakeholder $Role_T$ telling his story cannot continue after he hands over a critical artifact, requests information or starts any other form of interaction with another stakeholder $Role_{Req}$. Since $Role_T$ does not know what Desai et al. [5] refer to as local and usually *private policies*, which dictate how $Role_{Req}$ acts or reacts in a specific situation, we can never be completely sure what happens. We refer to such a potential dead end during the elicitation as a *stalemate* – without information on how another role continues the scenario, requirements engineers and the stakeholder can only guess what happens next.

Generally, a stalemate can only be overcome if the requirements engineers gather the other side of the story. Similar to a black box, we can only assume how $Role_{Req}$ continues after being triggered. Still, while $Role_{Req}$'s actions are not yet known, $Role_T$ can describe most of the possible outcomes, i.e., how he is affected by the result, based on experience. Similar to a jigsaw puzzle, many pieces of information exist, however, only few of them are required to complete individual scenarios. Thus, it is essential that individual stakeholders do not give up at the first stalemate, but are able to continue to describe their expectation(s) as well as their follow-up actions. Even if a stalemate occurs during an elicitation of a scenario, a stakeholder can still answer the questions in Table I.

Q_1 provides the requirements engineers with the information of who to talk to next to complete the scenario. Only a stakeholder identified as role $Role_{Req}$, i.e., someone who usually receives $Role_T$'s request, knows how to continue. For the Notifier, this would be the CommunicationsOperative (ComOp) who he notifies about an emergency (Figure 6). After this operative passed on the information, the Notifier expects to hear from her again – consequently, the Q_2 would be answered the same. Still, in other cases, the person being interacted with is not the one who responds. To be able to distinguish between different responses from different stakeholders, Q_2 provides information on who else might provide a response $Role_T$ expects.

The simulation has a specific state s_m, in which the stalemate occurred. In this state, an interaction has been started that results in a change for $Role_T$ – although he does not know how anyone else might be affected as a side effect, the stakeholder can still describe what changes for him (Q_3). Since stakeholders can only describe changes that affect them directly and that are visible for them, the expectation can only be a partial state based on the perspective of an individual stakeholder. In the example provided in Table I, the Notifier expects to receive an estimation on when someone will arrive. Thus, using the vocabulary already established as part of the domain model $\mathcal{D}_W$, this expectation can be described explicitly.

IV. EXPECTATION TRIGGERS & FOLLOW-UP ACTIONS

We define an *expectation* as the partial state a role expects to perceive between a pair of states s_m and s'_m. Since a single stakeholder cannot know how the overall state of all stakeholders changed in-between, he can only specify his perspectives of the respective states. Thus, in case of the Notifier, he expects $s_m|_{Notifier}$ and $s'_m|_{Notifier}$ to be identical, except that he has to receive an EstimatedArrivalMessage from a CommunicationsOperative. Whether a boat already departed to his location or whether another emergency happened somewhere else is unknown to the Notifier, as long as none of those things are visible to him. The expected follow-up state can be represented and described in different ways to be suitable for stakeholders. While Figure 7 (*right*) illustrates it as a partial state in an object diagram, Figure 5 (*right*) presents a natural language representation that can easily be understood and modified: *[CommunicationsOperative] sends [EstimatedArrivalMessage] to [you (Notifier)].*

In Section III, a stalemate s_m occurred and the participating stakeholder representing a Notifier was asked to verbalize his expectations on how another stakeholder he interacted with might respond or, more generally, how his context might change. From his answers to the questions in Table I, a trigger for the other stakeholder can be generated. A *trigger* is a tuple (s_{sm}, $Role_T$, $Role_{Req}$, $Role_{Resp}$, $s'_{sm}|_{Role_T}$). It contains the *stalemate* state s_{sm} as it occurred during the simulation. Further, the role of the participant who defined the trigger ($Role_T$) is included. Additionally, to resolve the trigger, it is essential to know, which role is expected to continue the scenario and which role directly interacts with $Role_T$ next ($Role_{Req}$ and $Role_{Resp}$, respectively). Finally, the partial state $s'_{sm}|_{Role_T}$ that $Role_T$ expects to observe afterwards is included as well. Based on the answers of Notifier in Section III, the resulting

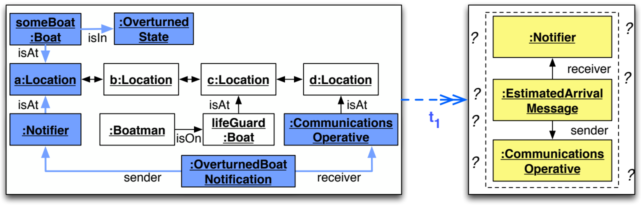
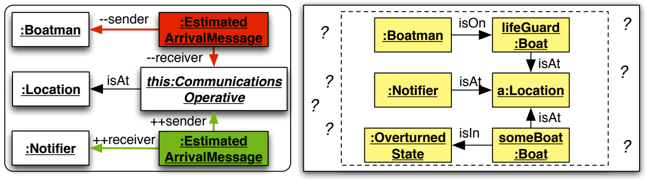


Figure 7. Based on s_m (*left*, Notifier's visibility is highlighted in *blue*), the Notifier can describe changes he expects as a partial state $s'_m|_{Notifier}$ (*right*) in $\mathcal{D}_W$'s vocabulary, leading to trigger t_1



(a) Story pattern x based on follow-up action f_I

(b) Eventually, the Notifier expects a lifeguard boat to arrive at his location

Figure 8. While the ComOp defines a follow-up action f_1, which leads to story pattern x (*a*), the Notifier expects to see a lifeguard boat (*b*).

trigger would be $t_1 = (s_m, Notifier, ComOp, ComOp, s'_m|_{Notifier})$, as illustrated in Figure 7.

A. Triggers and Alternatives

After the ComOp's *GoTo* message sending a Boatman to another location in response to the notification, ComOp expects that the Boatman responds with an *EstimatedArrivalTime* as prescribed by their protocol. Consequently, the ComOp defines a trigger $t_2 = (s_n, ComOp, Boatman, Boatman, s'_n|_{ComOp})$ expecting this message. While the ComOp might usually get an *EstimatedArrival* message, she might also be confronted with a *LowOnFuel* message, indicating that the boat needs to refuel first (Figure 9). Of course, the ComOp knows that all boats are fueled up at the beginning of each weekend. However, she does not know how much gas each boat may have left after several hours of service – an information only available to each respective Boatman. Thus, although a ComOp knows both possible outcomes, she cannot know, which one she will be confronted with, since she has no access to the information required for this decision.

From s_n, as for most stalemates, multiple continuations are possible from ComOp's point of view. Consequently, the *LowOnFuel* alternative can re-use s_n and is defined as trigger $t_3 = (s_n, ComOp, Boatman, Boatman, s''_n|_{ComOp})$ (cf. Figure 9).

B. Follow-Up Actions

A *follow-up action* f is an activity of a role, which is expected to apply if a specific precondition is fulfilled. It is characterized by a pair of states, the first being a precondition ($s_F|_{Role_T}$), which has to be fulfilled to execute the changes specified in the second ($s'_F|_{Role_T}$). As sketched in Table II, a follow-up action is the answer on how a role would continue after a stalemate has been overcome,

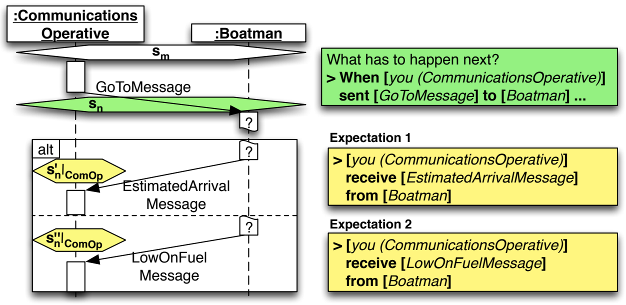

:Communications Operative
:Boatman
sm
GoToMessage
sn
?
alt
s'n|ComOp
EstimatedArrival Message
s''n|ComOp
LowOnFuel Message
What has to happen next?
> When [you (CommunicationsOperative)] sent [GoToMessage] to [Boatman] ...
Expectation 1
> [you (CommunicationsOperative)] receive [EstimatedArrivalMessage] from [Boatman]
Expectation 2
> [you (CommunicationsOperative)] receive [LowOnFuelMessage] from [Boatman]

Figure 9. Both expectations on how a Boatman may react to a *GoTo* message as experienced before and, thus, expected by a ComOp

TABLE II
BY ANSWERING THESE QUESTIONS, A STAKEHOLDER SPECIFIES A DISTINCT STATE AND HOW HE OR SHE FOLLOWS UP ON IT

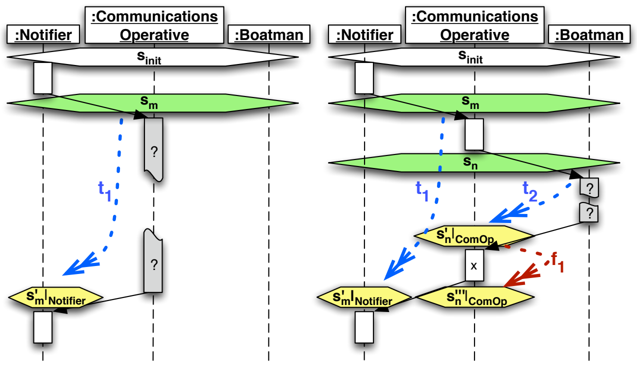
	Question for Stakeholder	*Answer of ComOp*	*Named*
Q_A	When do you become active?	"After the Boatman sent me an Estimated-ArrivalMessage"	$s_F\|_{Role_T}$
Q_B	How do you continue after $[s_F\|_{Role_T}]$?	"I send the Notifier an EstimatedArrivalMessage"	$s'_F\|_{Role_T}$

i.e., after the expectation has been fulfilled. After ComOp's expectations have been captured in t_2, the participating stakeholder can still describe how she as a ComOp would continue after her expectation ($s'_n|_{ComOp}$) is fulfilled. Consequently, $s'_n|_{ComOp}$ is presented to the stakeholder, either in an interactive visualization or in a textual representation. Based on this perspective, the stakeholder is able to specify the differences that her follow-up actions result in. In our example, the answer would be: *"I send the Notifier an EstimatedArrivalMessage"* ($s'''_n|_{ComOp}$, cf. Table II). Since the trigger has to be resolved for the stakeholder to follow up, the current state of the simulation needs to match the postcondition of the trigger. Thus, the postcondition of this trigger can be used as the precondition of the follow-up action. Combined with the follow-up state, this leads to the follow-up action $f_1 = (s'_n|_{ComOp}, s'''_n|_{ComOp})$. Similar to a pair of complete states, these two partial states can be used to derive a story pattern x (Figure 8*a*), which captures what the ComOp wants to achieve from her perspective.

In case of the ComOp, the precondition was already defined and was reused. If no trigger was defined beforehand, the stakeholder may still describe both situations, i.e., answer Q_A and Q_B, using natural language as sketched in Figure 5 (*right*). Of course, after having defined a follow-up action, additional follow-up actions can be defined.

V. RESOLVING TRIGGERS – SYNTHESIS

After the requirements engineer elicited an incomplete scenario ending in a stalemate s_{sm} and at least one trigger $t_m = (s_{sm}, Role_T, Role_{Req}, Role_{Resp}, s'_{sm}|_{Role_T})$, the next stakeholder to talk to is already predetermined. To complete this scenario, a stakeholder of the corresponding role $Role_{Req}$ needs to participate to continue the interaction with $Role_T$. The requirements engineer starts a simula-



(a) Incomplete scenario after a Notifier session (b) Same scenario after a ComOp follows up

Figure 10. The expectation described in t_1 *(a)* might be fulfilled after t_2 has been resolved and follow-up action f_1 was executed *(b)*

tion session by loading the state s_{sm} for $Role_{Req}$. The visualization of s_{sm} corresponding to $Role_{Req}$'s perspective should visualize the last interaction with $Role_T$ in such a way, that the participating stakeholder is able to identify, which scenario the requirements engineers are currently interested in. By explicitly loading s_{sm}, inconsistencies between different triggers and their continuations can be avoided, since all follow-up activities are direct responses leading towards the expectation $s'_{sm}|_{Role_T}$.

In a simple case, $Role_{Req}$ is also the responding role $Role_{Resp}$, which can answer directly and fulfill $Role_T$'s expectation immediately with a suitable response. In case of t_1, however, ComOp cannot yet fulfill Notifier's expectation ($s'_m|_{Notifier}$). During the simulation for t_1, all activities of ComOp ($Role_{Req}$ and $Role_{Resp}$) and any other role involved (Boatman) within the interactive visualization of s_m lead to a state s_x, in which ComOp responded as expected.

An expectation is fulfilled, if its partial state $s'_{sm}|_{Role_T}$ expected by $Role_T$ can be found in s_x. Since $s'_{sm}|_{Role_T}$ and s_x are structural specifications conforming to $\mathcal{D}_W$, the simulator can try to match the expectation $s'_{sm}|_{Role_T}$ to a part of s_x. If such a match can be identified, the context of $Role_T$ in s_x is as expected: $s_x \supseteq s_x|_{Role_T} \supseteq s'_{sm}|_{Role_T}$. Consequently, if $Role_{Resp}$'s activities led to such a state, these observed activities are a suitable continuation for the scenario from the point of views of $Role_T$ and $Role_{Resp}$.

To resolve the trigger t_1, that Notifier created in a first session (1st row in Figure 12), its stalemate s_m (illustrated in Figure 7) is loaded and the participating ComOp stakeholder receives Notifier's notification of an overturned boat in the corresponding visualization of s_m. As always, the ComOp continues by starting an interaction with a Boatman by sending a *GoToMessage*, thereby bringing the simulation into the state s_n (2nd row). At this point, the ComOp cannot continue to play-in what needs to be done, since a stalemate s_n is reached in which she cannot deterministically predict how the Boatman will react. As outlined in Section IV-A, two different scenarios are possible. After the ComOp defined

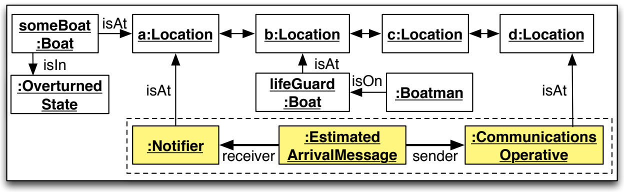


Figure 11. After ComOp is simulated using the story pattern x, the simulation is in state s_x with $s_x \supset s'_m|_{Notifier} = s'''_n|_{ComOp}$ (fulfilling the highlighted expectation in Figure 7)

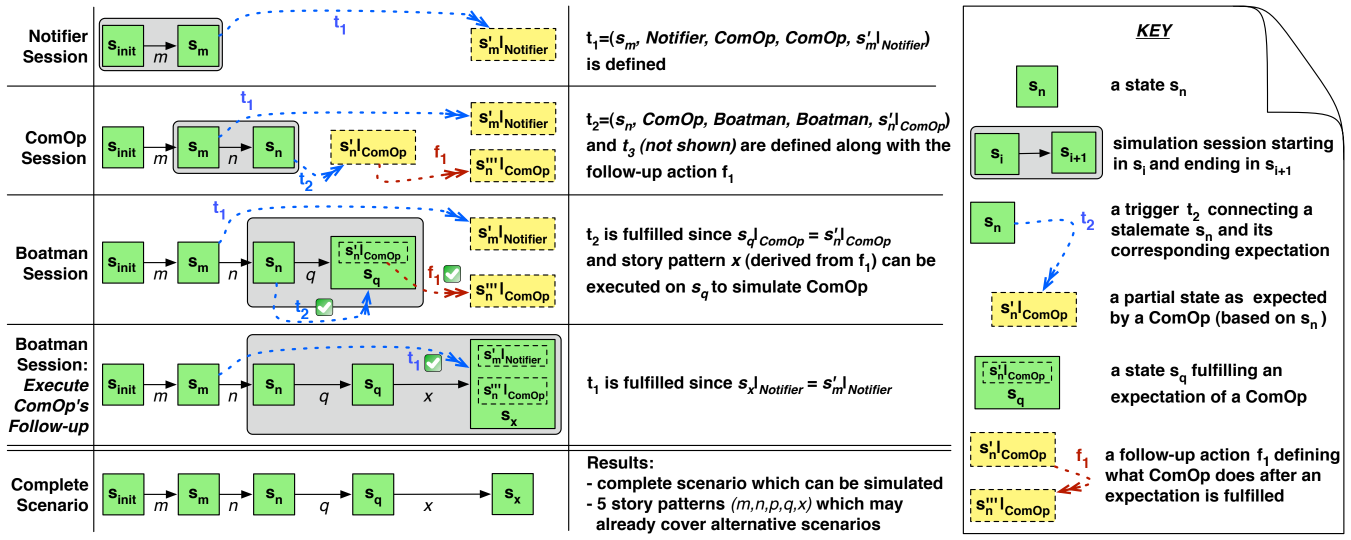
both corresponding expectations (t_2 and t_3), she also defines the follow-up action f_1 (cf. Section IV-B), which leads to the story pattern x (cf. Figure 8a) of how she continues after the expectation of t_2 has been fulfilled.

As specified in t_2 and t_3, the next role to talk to is Boatman, who needs to continue from s_n. After the stakeholder reviewed s_n in his visualization, he responds with an estimated arrival time (3rd row), thereby leading the simulation into state s_q in which Boatman fulfilled ComOp's expectation as defined in t_2. In s_q, ComOp's follow-up action f_1 is applicable since its precondition is identical to t_2's postcondition ($s'_n|_{ComOp}$). Consequently, by resolving t_2, the story pattern x is executed to simulate ComOp's follow-up action, leading the simulation into the follow-up state s_x (4th row). More importantly, the initial expectation of Notifier, i.e., the partial state $s'_m|_{Notifier}$, can be matched since the expected answer was provided through ComOp's follow-up action (cf. Figure 11). Thus, the requirements engineers end up with a completed scenario (5th row). Also, the story patterns necessary to reproduce and simulate it for other stakeholders can be derived from observations or follow-up actions. Now, the requirements engineers can continue by collecting alternatives, e.g., *what has to happen when a Boatman fulfills the expectation described in t_3?*

All triggers that are collected along the way are stored next to the story patterns, which are derived from observed scenarios. To resolve them, the simulator simply checks whether the expected outcome of an interaction ($s'_{sm}|_{Role_T}$) can be matched in the current state s_i of the simulation. Based on this algorithm, scenarios are completed step by step from stakeholder to stakeholder as illustrated for the notification example in Figure 12. The possibility remains, that no state $s_x \supseteq s'_m|_{Notifier}$ can be reached – even after multiple sessions of the role that is expected to reply. In this case, the requirements engineer has to be notified and two options are present: directly intervene and either talk to $Role_{Resp}$ to ask, e.g., *what needs to be true for the Boatman to not answer as expected* or talk to Notifier ($Role_T$) to check whether the expectation that was described is correct.

VI. Related Work

One of the main contributions of Harel and Marelly's Play-approach [11], [12], is the possibility not only replaying captured system behavior (play-out), but the possibility to capture additional system behavior (play-in) and, thus,



t_1=(s_m, *Notifier, ComOp, ComOp*, $s'_m|_{Notifier}$) is defined

t_2=(s_n, *ComOp, Boatman, Boatman*, $s'_n|_{ComOp}$) and t_3 *(not shown)* are defined along with the follow-up action f_1

t_2 is fulfilled since $s_q|_{ComOp} = s'_n|_{ComOp}$ and story pattern *x* (derived from f_1) can be executed on s_q to simulate ComOp

t_1 is fulfilled since $s_x|_{Notifier} = s'_m|_{Notifier}$

Results:
- complete scenario which can be simulated
- 5 story patterns *(m,n,p,q,x)* which may already cover alternative scenarios

a state s_n

simulation session starting in s_i and ending in s_{i+1}

a trigger t_2 connecting a stalemate s_n and its corresponding expectation

a partial state as expected by a ComOp (based on s_n)

a state s_q fulfilling an expectation of a ComOp

a follow-up action f_1 defining what ComOp does after an expectation is fulfilled

Figure 12. After only three stakeholder sessions, the scenario has been completed with two triggers and one follow-up action

new scenarios while doing so. Their approach, however, is centered around user interfaces – for each input the user provides, she can play-in how the system should react. While this might suffice to capture the interaction between individual stakeholders and a software system, i.e., its user interface (UI), the elicitation of interactions between different stakeholders is more complex.

Still, the play-engine [12] can be used to enable stakeholders to perceive the complete state the system is in, reduced to what is presented in the UI. The similar visualization approach is used by Ponsard et al. for specific goals such as whether a door of a train is closed when its moving [21] or even to represent domain-specific UI elements and how they affect each other from the point of view of a specific stakeholder [16]. However, these approaches are limited to single stakeholders and their interaction with a software system only – collaborative processes with information asymmetry cannot be elicited or validated.

Scenario-based approaches have been researched quite broadly, most notably by Uchitel et al. [18][19][20]. Starting with sets of potentially incomplete, implied or negative scenarios, they are able to derive state machines for the involved components that are suitable for all of these scenarios, in case of Whittle et al. even hierarchical ones [23]. Still, these approaches are not suitable to elicit human interactions, which are limited by what stakeholders can perceive from their individual context. To obtain behavioral models of what the stakeholders do, it does not suffice to know, which messages arrived at a stakeholder in which sequence, but rather, which artifacts and information they can see or access. For instance, the Notifier's expectation, that a lifeguard boat arrives (Figure 8*b*) can only be fulfilled by the Boatman moving to the corresponding location (Figure 4*b*). Only by eliciting and validating them in a state-based manner, it becomes viable to visualize the current *state* a stakeholder is in based on what she has access to.

Similar to our approach, Ghezzi et al. [10] compare pairs of succeeding states to derive the behavior of Java classes as graph transformations. While their method of automatically eliciting behavior is identical, their approach is restricted to a single actor, i.e., an instance of a class, and cannot cope with information asymmetry. The same goes for van der Aalst's ProM approach [22], which is able to derive a process model of how people can achieve their goals collaboratively based on log files detailing different scenarios. However, ProM is only able to use the information that is available in these logs, provided a logging system is already in place and analog interactions cannot be captured.

VII. Conclusion and Future Work

In this paper, we presented an approach that reduces the effort necessary to elicit and validate new collaborative scenarios. Thereby, the number of sessions necessary to elicit a stakeholder's actions within a scenario can be reduced. In case of stalemates, stakeholders can express their expectations on interaction results in form of partial states. Based on these partial scenarios, stakeholders are then able to play-in continuing behavior decoupled from one another. Our simulator synthesizes the captured individual perspectives to obtain the complete scenarios, thereby overcoming the inherent fragmentation of different perspectives. We discussed how this technique extends our model-based validation approach to be applicable for model-based elicitation, too. The method was illustrated on a real life example of a lifeguard service.

Using the old approach, the requirements engineer would have to go back and forth between two or more stakeholders for each interaction. Thus, the total number of sessions required to elicit a complete scenario related to the number of stalemates the stakeholders ran into during the elicitation, since an additional session is necessary for each stalemate that occurred. Generally, the number of stalemates per role can vary strongly. Especially for the role ComOp in the lifeguard example, at least ten interactions need to be elicited for each scenario, which leads to the same number of sessions to resolve the implied stalemates. However, by being able to express her expectations and, thus, triggers intuitively, the ComOp is now able to defer the completion of all scenarios to other stakeholders later on. Consequently, instead of ten sessions to overcome ComOp's stalemates only, ComOp can express all her contributions to the collaborative scenarios in just one session. Decoupled from her, the other stakeholders can then complement the scenarios accordingly. By systematically completing these scenarios, the total number of elicitation sessions no longer depends on the number of interactions and stalemates, but on the number of roles and their ability to express their expectations. In this case, our approach can end up with only one elicitation session per role.

For future work, we want to evaluate whether stakeholders are able to describe all of their interactions in the proposed way and investigate, for which domains and scenarios this technique works best. Our approach currently focuses on completing one scenario at a time. We plan to investigate how the overall number of sessions can be further reduced by taking into account how stakeholders interact in multiple scenarios. For instance, the resolution of triggers from multiple scenarios might be handled in the same session.

References

[1] A. Al-Rawas and S. Easterbrook. Communication Problems in Requirements Engineering: A Field Study. In *Proc. of the First Westminster Conference on Professional Awareness in Software Engineering*. Royal Society, London, 1996.

[2] I. Alexander and N. Maiden, editors. *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley, New York, 2004.

[3] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer. *Software & Systems Requirements Engineering: In Practice*. McGraw-Hill, Inc., New York, NY, USA, 2009.

[4] A. Davis and K. Nori. Requirements, plato's cave, and perceptions of reality. *Computer Software and Applications Conference, Annual Int.*, 2:487–492, 2007.

[5] N. Desai, A. Mallya, A. Chopra, and M. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31:1015–1027, 2005.

[6] G. Gabrysiak, H. Giese, and A. Seibel. Interactive Visualization for Elicitation and Validation of Requirements with Scenario-Based Prototyping. In *Proc. of the 4th International Workshop on Requirements Engineering Visualization*, pages 41–45, Los Alamitos, USA, 2009. IEEE Computer Society.

[7] G. Gabrysiak, H. Giese, and A. Seibel. Using Ontologies for Flexibly Specifying Multi-User Processes. In *Proc. of ICSE 2010 Workshop on Flexible Modeling Tools*, Cape Town, South Africa, 2010.

[8] G. Gabrysiak, H. Giese, and A. Seibel. Towards Next-Generation Design Thinking II: Virtual Multi-User Software Prototypes. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking Research*, Understanding Innovation, pages 107–126. Springer, 2012.

[9] G. Gabrysiak, R. Hebig, and H. Giese. Simulation-assisted elicitation and validation of behavioral specifications for multiple stakeholders. In *Proc. of the 21st IEEE International Conference on Collaboration Technologies and Infrastructures*, Toulouse, France, June 2012.

[10] C. Ghezzi, A. Mocci, and M. Monga. Synthesizing intensional behavior models by graph transformation. In *Proc. of the IEEE International Conference on Software Engineering*, pages 430–440, Washington, USA, 2009. IEEE.

[11] D. Harel, H. Kugler, and A. Pnueli. Synthesis revisited: Generating statechart models from scenario-based requirements. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and Systems Modeling*, pages 309–324. Springer, 2005.

[12] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[13] R. Heckel. Graph transformation in a nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1):187 – 198, 2006. Proc. of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques.

[14] T. Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5:369–385, 2006.

[15] A. Luebbe and M. Weske. Tangible media in process modeling – a controlled experiment. In H. Mouratidis and C. Roland, editors, *23th Conference on Advanced Information Systems Engineering (CAiSE 2011)*, pages 283–298, 2011.

[16] C. Ponsard, N. Balych, P. Massonet, J. Vanderdonckt, and A. van Lamsweerde. Goal-Oriented Design of Domain Control Panels. In S. W. Gilroy and M. D. Harrison, editors, *DSV-IS*, volume 3941 of *LNCS*, pages 249–260. Springer, 2005.

[17] N. Seyff, N. Maiden, K. Karlsen, J. Lockerbie, P. Grünbacher, F. Graf, and C. Ncube. Exploring how to use scenarios to discover requirements. *Requirements Engineering*, 14(2):91–111, 2009.

[18] S. Uchitel, G. Brunet, and M. Chechik. Synthesis of Partial Behavior Models from Properties and Scenarios. *IEEE Transactions on Software Engineering*, 35(3):384–406, 2009.

[19] S. Uchitel, J. Kramer, and J. Magee. Detecting implied scenarios in message sequence chart specifications. *SIGSOFT Softw. Eng. Notes*, 26:74–82, September 2001.

[20] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13:37–85, 2004.

[21] H. T. Van, A. van Lamsweerde, P. Massonet, and C. Ponsard. Goal-oriented requirements animation. *Requirements Engineering, IEEE International Conference on*, 218–228, 2004.

[22] W.M.P. van der Aalst. Trends in business process analysis: From validation to process mining. In *International Conference on Enterprise Information Systems*, Funchal, Portugal, 2007.

[23] J. Whittle and P. Jayaraman. Synthesizing hierarchical state machines from expressive scenario descriptions. *ACM Trans. Softw. Eng. Methodol.*, 19:8:1–8:45, 2010.