Simulating User Journeys with Active Objects [⋆]

Paul Kobialka[1], Rudolf Schlatte[1], Gunnar Rye Bergersen[1,2], Einar Broch Johnsen[1], and S. Lizeth Tapia Tarifa[1]

[1] Dept. of Informatics, University of Oslo, Oslo, Norway
{paulkob,rudi,gunnab,einarj,sltarifa}@ifi.uio.no
[2] GrepS B.V., Utrecht, the Netherlands

Abstract. The servitization of business makes companies increasingly dependent on providing carefully designed user experiences for their service offerings. User journeys model services from the user's perspective, but user journeys are today mainly constructed and analyzed manually. Recent work analyzing user journeys as games enable optimal service-provider strategies to be automatically derived, assuming a restricted user behavior. Complementing this work, we here develop an actor-based modeling framework for user journeys that is parametric in user behavior and service-provider strategies, using the active-object modeling language ABS. Strategies for the service provider, such as those derived for user journey games, can be automatically imported into the framework. Our work enables prescriptive simulation-based analyses, as strategies can be evaluated and compared in scenarios with rich user behavior.

1 Introduction

Companies increasingly offer services to enhance their product range, a development termed the *servitization of business* [49]. The success of these services is highly dependent on user satisfaction: If the users are satisfied with how they experience the offered service, the companies are rewarded financially without increasing their risk [23]. Therefore, to provide successful services, companies need to adjust and improve their services from the *users'* perspective. However, services are usually analyzed from the *managerial* perspective, centered on the company and not on the users.

User journeys allow services to be analyzed from the user perspective, with the aim of understanding and hopefully improving the user's experience of a service. User journeys model a user's actual path through a service by capturing so-called touchpoints; these reflect communication between the user and the service provider, or actions performed by the user. Due to lacking formalization and tool support, the analysis of user journeys is today mainly a manual process [25]: analysts collect feedback on a service from a representative group of users (e.g., by means of questionnaires) to manually construct a user journey map, look

[⋆] This work is part of the *Smart Journey Mining* project (Research Council of Norway, grant no. 312198) and the SIRIUS Centre for Scalable Data Access (Research Council of Norway, grant no. 237889).

for typical pain points, and possibly suggest improvements to the user journey. Because the user journey analysis is manual, the process is not easily applied to complex services or analyzed with respect to many users.

The interaction between a service provider and a user can be formalized as a game [31, 32]. Users interact with the service provider to achieve a specific goal and the service provider may adopt different strategies to handle the users. Strategies for these games can be automatically analyzed using model checking tools such as UPPAAL [21] and PRISM [16] to reveal insights about the user journey. The analysis of user journey games can identify pain points in the user journeys (e.g., states where users abandon their journey), where the user journey could be improved. User journey games assume that all users are equally antagonistic, i.e., users always choose the worst possible action. However, in a real scenario users are not always uniformly antagonistic to the service provider, which makes it interesting to consider approaches that can relax this assumption.

In this paper, we propose to model user journeys by means of actors, to explore more diverse user behavior, complementing previous work on user journey games. We model a service provider and concurrent users as independent actors. The resulting actor model allows us to capture richer interaction scenarios between users and a service provider, and facilitates more realistic models than with user journey games, e.g., to explore several different service-provider strategies. Further, we consider parameterized and randomized user models to differentiate user behavior and explore the effect of service-provider strategies under different assumptions about such user behavior. Specifically, we investigate a user compliance parameter expressing the probability of a user waiting for the service provider's guidance instead of just taking a random action (e.g., the willingness or capability of users to follow instructions). Our model supports prescriptive analysis of user journeys by varying service-provider strategies and comparing the consequences of strategic decisions in the user journey.

We implement the user journey model as an actor-based simulation framework, using the active objects of ABS [27, 29]. ABS is a timed actor-based modeling language, which supports cooperative scheduling and the specification of timing- and resource-aware behavior. Cooperative scheduling allows a process, executing in an actor, to be suspended while waiting for an event to occur, such that another process that is able to make progress can execute. Timed semantics allow the specification of the temporal behavior in the model. Resource-aware behavior takes a supply-and-demand perspective of execution, relating locations that provide resources to actors that require them for executing their active processes and modeling part of a system that has limited resources.

In this paper, we focus on the development of the core framework using ABS constructs without time and resources. We envision exploiting the time and resource aspects of ABS to reveal the bottlenecks of the service due to the waiting times of users and limited resources in the service (e.g., waiting for telephone calls or manual checks in the service). Towards this aim, we now focus on capturing the functional aspects of our proposed actor framework, which is parametric in both user behavior and the service provider's strategy. For example, service provider strategies derived using the above-mentioned model checking techniques can be automatically imported into our simulation framework.

Recent extensions to the ABS simulation tool [42], implemented in Erlang [3], allow the parameters of the framework to be instantiated in a data-driven way by means of SQL queries to instantiate user behavior and service-provider strategies into user-defined datatypes in ABS, that later can be used to drive the execution of the model. We then use simulations to conduct experiments on the resulting user journey model for different user parameters, i.e. we investigate user journeys for varying probabilities of user compliance on randomized users. We evaluate our actor model of user journeys on an industrial case study; the results are reviewed by a long-term employee of the cooperating company, who is also the third author of this paper.

In short, the contributions of this paper are:

1. an active-object model for user journeys that is parametric in user behavior and in service-provider strategies,
2. a data-driven simulation framework to evaluate and compare different strategies for the service-provider, and
3. an application of the simulation framework to an industrial case study.

2 Motivating Scenario

Consider an imaginary company TestMe ltd. that offers evaluations of programming skills. Companies searching for new developers commission TestMe to conduct tests of their applicants to determine their level of programming skills. TestMe is paid per user (i.e., a user is here an applicant to the commissioning company) that completes the evaluation and does not withdraw in the middle of the evaluation process. Therefore, TestMe wants to investigate the user experience when users engage in the tests of the evaluation process and hires a team of analysts to analyze the user journey. The analysts start by conducting questionnaires with selected users and manually generate, based on the answers, a so-called user journey map outlining the experiences and feedback from the questionnaires. The user journey map may reveal pain points in the user journey, i.e., interactions hindering a successful completion of the skill evaluation.

To improve the user journey and engage the user more actively, TestMe may consider different changes in the evaluation process based on the information gained from the user journey analysis. Further, the company would like to differentiate the user journey analysis depending on the users' skill level, assuming that users at different skill levels behave differently during the evaluation process. To facilitate a continuous evaluation of user journeys, the analysts need an automated process of user journey analysis that does not depend on the manual processing of questionnaires. To address this bottleneck, previous work by the authors proposes the use of recorded logs from the system to automatically generate a model of the user journey, called a *user journey game* [32]. This approach drastically reduces the time until realistic models are available.

User journey games and strategies that ensure (or increase the chances for) a successful outcome of these games are introduced in Section 4.1.

The user journey games can be used by the team of analysts at TestMe to derive (winning) strategies suitable for the service provider, i.e., strategies that guide users toward completing the evaluation. However, the analysts struggle with the strict assumptions in user journey games, needed for successful analysis. User journey games do not distinguish users with, e.g., different skill levels, preventing the desired prescriptive analysis based on different users. To overcome these limitations, we here propose to model user journeys using active objects in the Abstract Behavioural Specification (ABS) language [27] (ABS is summarized in Section 4.2), and integrate the strategies derived from user journey games in an active object setting. The resulting workflow is outlined in Figure 2.

Sections 5.1 and 5.2 discuss how to model user journeys as active objects (*Step 1* in Figure 2), where we describe the transfer from Uppaal [34] models to ABS and the intermediate steps needed to encode generated strategies. Section 5.3 introduces parameterized user behavior to differentiate users and expands on the model generation. The model is further specified with additional user parameters so that assumptions needed for games are removed (*Step 2*). Further, we simulate different kinds of users to evaluate possible changes to the service provider's behavior (*Step 3*). Our simulation model is parameterized in the user behavior and allows adaptations to reflect different user behavior, corresponding to the different kinds of users encountered by the company. Section 6 describes the conducted simulations and evaluates our approach on a real case study. We summarize our work in Section 7 and outline future work.

3 Related Work

We discuss related work with respect to the data-driven analysis of user journeys and the modeling capabilities provided by the active object language ABS. To the best of our knowledge, this is the first work on modeling user journeys in an actor or active object language, giving all actors an operative role.

User journeys express the interactions between a service provider and its users from the users' perspective [22,46]. Various modeling notations have been proposed to support the blueprinting process [13], establishing a model of the planned interactions between the user and service provider for a service. Approaches to create user journey diagrams include [5,18,26,33,39,40]; in most of these approaches, diagrams are created by hand after conducting surveys and questionnaires. Digital support exists in e.g. [33] to visualize static information of the interactions such as the time spent from the user's perspective, the experience per interaction, etc.

The *Customer Journey Modeling Language* (CJML) [24, 26] offers two diagram types to highlight different aspects of the users' perspective: *customer journey network diagrams* display the interaction between the user and all subcontractors, *customer journey diagrams* display the impressions from the users' point of view. CJML highlights deviations in the actual journey, the actual impressions a user has in the service, from the planned journey, the planned impressions. The *Smart Journey Mining* project aims to build data-driven tool support for user journeys [25]. Therefore, CJML was actively extended for digital support; CJML v2.0 provides an XML format for the in- and export of actual and planned user journeys [26].

Data-driven methods from process mining [1] for process discovery have been successfully applied to discover user journeys from recorded logs. Bernard *et al.* [8, 10] investigate the possibility of using process mining for user journeys, they use hierarchical clustering and user-defined goals to abstract from a large number of journeys [7], and propose a method to discover user journeys from logs at varying levels of granularity [9]. Terragni and Hassani [44] investigate user journeys in the form of web logs and their optimization by building recommender systems proposing user-specific actions optimizing key performance indicators [45]. In contrast, our work focuses on the modeling aspect of user journeys with active objects and simulations to gain prescriptive insights into the service provider behavior and user journeys.

Formal methods allow the verification and analysis of discovered models for desired properties. David *et al.* present TAPAAL [19], a model checker for timed-arc Petri nets, which has been used by Bertolini *et al.* [11] to verify requirements in the healthcare domain. Kobialka *et al.* [31, 32] proposed *user journey games* as a formal model for user journeys, where the user and service provider are independent actors competing for a successful user journey. In [31] the approach is applied to a large process mining benchmark log and a state reduction method on event level is proposed.

Challenges for leveraging formal, compositional language semantics to industrially applicable tools, including how to input/output real-world data, have been discussed in the context of ABS in [43]. The simulation tool of ABS has previously been used to model and analyze large use cases (e.g., [2,12,30,36,37,42]); in particular, Turin *et al.* [47] use ABS to build and analyze a formal model for cloud deployment in Kubernetes, illustrating the impact of large loads of users.

4 Preliminaries

4.1 User Journeys as Weighted Games

A game [4, 15, 16, 21] consists of players that alternate in deciding on the action to take as the game transitions from one state to the next. Players may have strategies to try to force a specific outcome of the game; e.g., a player may try to reach a desired outcome of the game or to ensure that certain states are never reached. Actions in a game can have weights, e.g., to express rewards or penalties when taking an action, transforming the game into a weighted game.

A *weighted game* [15] is a tuple $(\Gamma, A_c, A_u, E, s_0, T, w)$ with a set Γ of states, sets A_c, A_u of *controllable* and *uncontrollable* actions (or labels) with $A_c \cap A_u = \emptyset$, a transition relation $E \subseteq \Gamma \times A_c \cup A_u \times \Gamma$, an initial state $s_0 \in \Gamma$, a set $T \subseteq \Gamma$ of final states, and a weight function $w : E \to \mathbb{R}$ that assigns weights to transitions. When analyzing a two-player game in which one player (the controller) takes controllable actions and the other player takes uncontrollable actions, it is assumed that only the controllable actions in A_c can be selected by the analyzer — the actions in A_u are nondeterministically decided by an adversarial environment, playing against the controller. If both players have actions available, the uncontrollable actions have precedence over the controllable actions.

In *user journey games* [32], the service provider and user are modeled as players in a two-player game, each with their own set of actions. Formally, a user journey game is a weighted game $(\Gamma, A_c, A_u, E, s_0, T, T_s, w)$, where $T_s \subseteq T$ are the successful final states. By using games as the user journey model, we inherently assume that (1) no player performs more than one activity concurrently, and that (2) user journeys are goal-driven processes where the user and service provider have the incentive to achieve the journey's goal, i.e., to reach a successful final state. For a user-centric analysis, the user is modeled as the adversarial environment that takes uncontrollable actions and the service provider as the controller that takes controllable actions. We require that the service provider has suitable responses for all user interactions and does not constrain the user.

The weights in user journey games reflect the users' experience as reflected in the system logs in the following way: interactions that only occur in successful journeys receive a positive weight, interactions that only occur in unsuccessful journeys receive a negative weight, and interactions that occur in both successful and unsuccessful journeys receive a neutral weight. The sum of weights along a (partial) user journey is called *gas*, and reflects the aggregated experiences of the respective users. In the games, a unique start state is introduced to ensure that all users start from the same state, and positive and negative final states are introduced to differentiate successful from unsuccessful journeys.

User journey games are generated from logs by (1) mining a transition system from the traces in the log, (2) transforming the transition system into a game by defining *controllable* and *uncontrollable* actions, and (3) adding user feedback by computing weights on the transitions. An entropy-based function assigns high positive weights to actions that primarily occur in successful journeys, high negative weights to actions that primarily occur in unsuccessful journeys and neutral weights to actions in successful and unsuccessful journeys. The generation of user journey games from event logs is detailed in [32].

A *strategy* [20] assigns a set of possible actions to every state in a game. Formally, given a game $G = (\Gamma, A_c, A_u, E, s_0, T, w)$, a strategy for G is a partial function $\sigma : \Gamma \rightarrow 2^{A_c \cup \{\lambda\}}/\{\emptyset\}$ from states in Γ to the power-set of controllable actions A_c; here, λ denotes the "wait" action (i.e., no controllable action is taken and the controller gives the next move to the environment) and the possibility of "no action" (expressed by $\{\emptyset\}$) is removed. We say that a player follows a strategy σ if, in every state $s \in \Gamma$, the player only selects actions in $\sigma(s)$. If there is a strategy that guarantees a desired property, the controller can enforce the desired outcome by following this strategy, preventing the adversary from making a choice that violates the property.

We here consider *memoryless* strategies, i.e., strategies where the choice of the next action only depends on the last state. Maler *et al.* [38] have shown that memoryless strategies suffice for reachability properties. Note that strategies can be nondeterministic; i.e., there might be more than one possible action available to enforce the desired outcome. We call a strategy *deterministic* if only one possible action can be selected in any state (i.e., $|\sigma(s)| = 1$ for all $s \in \Gamma$).

UPPAAL STRATEGO [21] is a model checker for games in the UPPAAL tool suite [34], which combines UPPAAL TIGA [4] with the stochastic model checker UPPAAL SMC to stochastically model check games; i.e., it verifies properties in a game setting through random simulations and hypothesis testing until sufficient statistical evidence is reached. UPPAAL STRATEGO allows refining a strategy towards an expected goal, e.g., to find the shortest path to a successful final state [20]. UPPAAL STRATEGO constructs strategies for adversarial users. When refining or evaluating strategies with respect to numerical criteria, e.g. estimating the expected number of steps in a user journey under a certain strategy, UPPAAL STRATEGO uses stochastic simulations.

4.2 The ABS Modeling Language

The *Abstract Behaviour Specification* [27] language (ABS) is a language for behavioral modeling of distributed systems. ABS is an active object language [14], combining executable actor-based semantics with asynchronous method calls and first-class futures. Data is modeled via a functional, side-effect-free layer of algebraic data types and parametric functions. The actor behavior is expressed in a sequential, imperative way, with explicit suspension points for cooperative scheduling in each actor. ABS has a Java-like syntax and is supported by a range of analysis tools (see, e.g., [41,50]). The internal state of each actor can be modeled in detail or completely abstracted, depending on the purpose of the model. The following features of ABS are useful in creating behavioral models:

Asynchronous method calls and first-class futures: The essential feature of a distributed system is that communication (sending a method call) and execution (scheduling an incoming call) are decoupled. The caller can continue execution until the result of a call is needed, and the callee can schedule calls from multiple callers as needed.

Process suspension and boolean guards: Inside an ABS actor, many processes can execute in a cooperative manner, with only one process running at any given time. Processes suspend themselves when waiting for a method call result or waiting for a boolean condition over the actor state.

Data Structures and Functions: Algebraic datatypes are used to model actor state and data that is passed between actors via method calls. Functions that are calculated over such datatypes are side effect-free.

Database Access. For the work presented in this paper, we use the recently added capabilities of ABS to import structured data stored in a SQLite database file into a running ABS model.

Structured data stored in an SQLite database can be directly read into ABS by converting query results into ABS datatypes. Executing a query inside ABS

```
data StrategyEntry =
  StrategyEntry(String strategy_state, String strategy_action);

def List<StrategyEntry> strategy(String strategy_name)
    = builtin(sqlite3, "../data/journeys.sqlite",
        "SELECT state, action FROM strategies WHERE strategy_name = ?",
        strategy_name);
```

Fig. 1: Querying the "journeys.sqlite" database from within ABS, passing in an ABS value as query parameter.

SQLite return value	**ABS**	**SQLite query parameter**
INTEGER	Int	INTEGER
INTEGER or REAL	Float	REAL
INTEGER or REAL	Rat	REAL
INTEGER (`0` = `False`, otherwise `True`)	Bool	`0` or `1`
TEXT	String	TEXT
Row of the above	User-defined datatype	n/a

Table 1: ABS to SQL datatype mapping: the first and second columns show the SQL result to ABS datatype conversion; the second and third columns show how ABS datatypes are converted into query parameter values.

produces a list of ABS data, which can be used like any other list after the query has finished. If the query only returns rows of one value each, e.g. `String`, the type of the query result inside ABS will be `List<String>`. If, on the other hand, the query returns tuples containing more than one value, the query will name the ABS datatype that holds each resulting row. The constructor of this ABS datatype has to accept parameters of the same number and type as returned by the query. For exampe, the result from a query like `SELECT name, age FROM persons`, which returns `(string, integer)` tuples can be stored in an ABS datatype defined like `data Person = Person(String, Int)`. Table 1 shows how SQL results are mapped to ABS values, and how ABS query parameters are mapped to SQL values.

Figure 1 illustrates how to import data into ABS from an SQLite file. For this example, let us assume that various strategies for a user journey game have been stored in the file `journeys.sqlite` containing entries that relate `strategy_name`, `state`, and `action` (See Section 4.1). It is possible to query such a file such that the records are stored in a list of strategy entries. In this example we define in ABS a datatype `StrategyEntry` that holds one entry from one strategy, and the function `strategy` that reads one full strategy from the SQLite table and stores it into a list `List<StrategyEntry>`.

Queries into the SQLite database can be parameterized in the standard way: parameters inside the query string are denoted by a question mark (`?`); values for these parameters are supplied as additional arguments to the query.

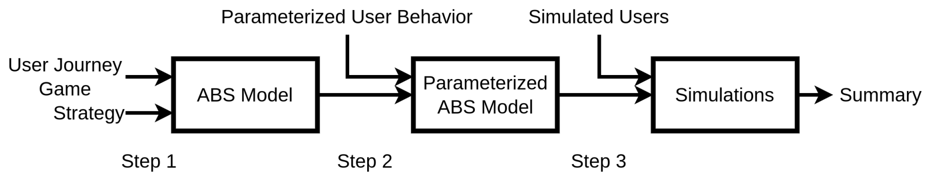


Fig. 2: Workflow pipeline.

Only basic datatypes (string, integer, float) can be supplied as parameters. The `strategy_name` parameter to the `strategy` function in Figure 1 is used as such a query parameter; its value ends up in the corresponding `WHERE` clause in the SQL query sent to the database engine.

5 Workflow Pipeline

We now consider a pipeline for analyzing user journeys by means of simulations of an active object model of user journeys. The pipeline is depicted in Figure 2 and consists of the following steps:

– *Step 1:* An ABS modeling framework imports user journey games and strategies from a database;
– *Step 2:* The model is adjusted by instantiating parameterized user behavior and modifying transitions to `finNeg` to be uncontrollable; and
– *Step 3:* Simulations are used to explore aspects of the user journey for given strategies of the service provider.

We develop an ABS model that implements users and service providers as active objects that communicate and run in parallel with each other. Additionally, a `WorkflowProvider` class that wraps all knowledge about strategies and available controllable and uncontrollable actions, serves as common knowledge base for both users and service providers. The model is parameterizable wrt. strategy, user behavior, and number of users. The output of a model run is the number and type of users in each final state, together with the average journey length and accumulated gas.

Generated games and strategies (see Section 4.1) are aggregated in an SQLite database that can be read from within ABS (see Section 4.2). In particular, strategies for user journey games can be generated from user journey games using Uppaal Stratego and integrated in the ABS model to guide users in simulations. Since the generated strategies are memoryless, they can thus be exported as a mapping from states to actions. Refining a strategy corresponds to refining the mapping to be deterministic, i.e. there is at most one suggested action per state.

We now explain how to prepare data that can be imported into the ABS model in Section 5.1, then how the ABS model is constructed in Section 5.2.

Source State	***Action***	***Target State***	***Controllable***	***Cost***
start	Registered	Registered	False	-1.9
start	AssignInstance	AssignInstance	True	-22
Started	TaskEvent	TaskEvent - 0	False	-2
ResultApproval	ResultsAccepted	ResultsAccepted	False	18

Fig. 3: Tables imported into the ABS model: The transition system as a list.

5.1 Data Preparation for the Workflow Pipeline

In Step 1 of the workflow, we import user journey games and strategies into ABS. The workflow produces a single database file that contains all necessary information to simulate different scenarios.

The user journey game is transformed into a CSV format, that enumerates states and available actions in each state, as a series of entries (source state, action, target state, controllable or uncontrollable, cost) that are imported into a database, see Figure 3. We export strategies from Uppaal Stratego 10 by using export queries:

```
saveStrategy("strategy.xml", strategy).
```

Strategies are then also transformed into tabular CSV format, mapping states to actions, see Figure 4, and we import the tables into the same SQLite database. Both imports cover Step 1 in Figure 2. In the *start* state, the company assigns a virtual instance to the user, *AssignInstance*. When it is *Started*, the company has to wait for the user to work on the *TaskEvent*, expressed as a *Wait* action in the strategy and an uncontrollable action in the process model.

We adapt the user journey game to run simulations where users can give up in the middle of their journey (and hence, reach the unsuccessful final state). In the imported user journey game, actions leading to the unsuccessful final state were defined as controllable; otherwise, model checking could never guarantee to reach the successful final state (cf. Section 1). This restriction is not needed for simulation; in the adapted version, actions leading to the unsuccessful final state are modeled as uncontrollable actions. These adaptations cover Step 2.

Source State	***Action***
start	AssignInstance
Started	Wait
ResultApproval	Wait

Fig. 4: Tables imported into the ABS model: The strategy as a state to action mapping.

Step 3 uses simulations to explore the model and possible service provider strategies. We simulate scenarios that combine the parameterized model with different users. In the ABS modeling framework, the model of the service provider and the user models are kept separate for easier construction and utilization.

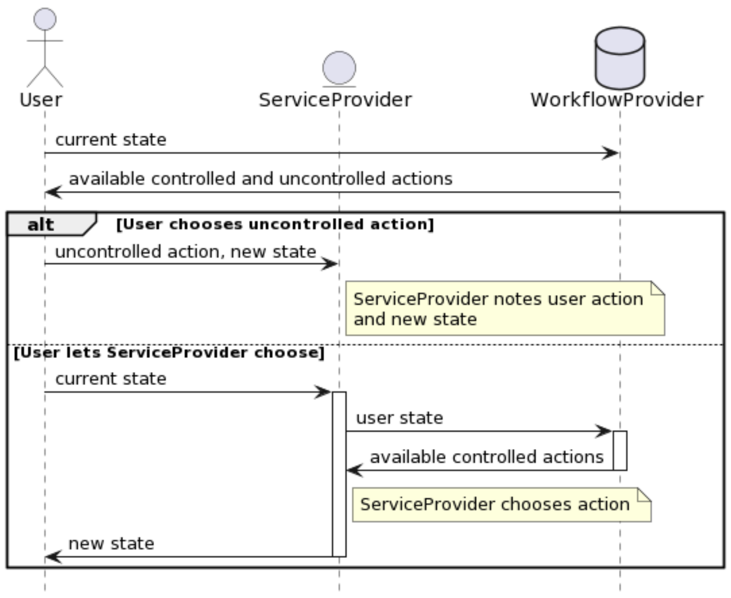

User
ServiceProvider
WorkflowProvider
current state
available controlled and uncontrolled actions
alt
[User chooses uncontrolled action]
uncontrolled action, new state
ServiceProvider notes user action and new state
[User lets ServiceProvider choose]
current state
user state
available controlled actions
ServiceProvider chooses action
new state

Fig. 5: Sequence diagram of one interaction in the simulation.

5.2 Modeling the User and the Service Provider

In the ABS model, the structure of the underlying user journey game is implemented via a component called *WorkflowProvider*, which is consulted by the user and the service provider objects. The ABS model contains one interface for users, one for service providers, and one for the workflow provider. Figure 5 shows the exchange of messages between the actors in the simulation.

Figure 6 shows the interfaces and data types of the simulation. As shown in the interaction diagram, the user is the "active" participant that initiates each round of choosing between actions. Consequently, the `User` interface offers no methods to be called from outside.

Since both the user and service provider need knowledge about the user journey game, the model encapsulates this knowledge in a common interface `WorkflowProvider`. Its method `available_tasks` returns all available controllable and uncontrollable actions, given a user identifier and the user's state. (The user identifier may be used to implement per-user strategies.)

The user chooses whether to perform a controllable or uncontrollable action and informs the service provider about its decision (see Figure 5). The `notifyUncontrolledAction` method notifies the service provider about the chosen action and new state of the user. In contrast, the `performControlledAction` method leaves the decision of the action to be taken to the service provider, which in turn consults the workflow provider about its options. The chosen action is returned to the user, who updates its state accordingly.

```
interface User { }

interface ServiceProvider {
    Unit notifyUncontrolledAction(
      Int user_id, String uncontrolled_action, String new_state);
    Maybe<WorkflowTask> performControlledAction(
      Int user_id, String current_state);
}

interface WorkflowDriver {
    WorkflowTasks available_tasks(Int user_id, String state);
}

data WorkflowTasks = WorkflowTasks(
    List<WorkflowTask> controllable_tasks,
    List<WorkflowTask> uncontrollable_tasks);

data WorkflowTask = WorkflowTask(
    String origin_state, String target_state,
    String action, String controllable, Float cost);
```

Fig. 6: The internal structure of the workflow simulation model.

The classes implementing these interfaces can be seen in the online repository,[1] which features the implementation of the workflow pipeline. The main variability is located in the implementations of the `WorkflowProvider` interface, where the modeler can set up workflow descriptions with varying strategies, or no strategy at all. The user class is parameterized with the likelihood of performing an uncontrollable action if applicable. The service provider class relies on the workflow provider for most of its behavior, but can be extended to implement resource-sensitive behavior (see the discussion of future work in Section 7).

5.3 Parameterized User Behaviour

User journey games aggregate the behavior of several users into one model, thereby assuming that all users are equally antagonistic; i.e., all users in the same state have the same available actions, and, when the service provider and the user both have available actions, all users have higher precedence than the service provider. These assumptions are captured in the strategies generated by Uppaal Stratego: antagonistic users exploit their precedence over the service provider when selecting the next action, and two different users in the same state can not be differentiated.

In reality, users differ based on individual properties which are abstracted away in user journey games. In our simulation framework, we would like the

[1] https://github.com/smartjourneymining/abs_journeys_aol-23/releases/tag/AOL23

user model to capture structural differences between users that are not expressed through choices in the user journey game but are determined already at the beginning of the game.

For this purpose, we let the user model have *parameterized user behavior* by introducing a parameter to the user model that is unknown to the service provider but fixed at run-time, i.e. for every instantiated user. This user parameter p ranges from $[0, 1]$ and models the probability that the user waits for the service provider's guidance; with probability $1 - p$ the user takes an uncontrollable action. This way, the parameter models the "compliance" of the user, changing the probability to wait for the service provider's actions or taking an arbitrary, uncontrollable transition. A non-compliant user, always taking uncontrollable actions, can be expressed with $p = 0$. A compliant user can be expressed with $p = 1$, waiting for the service provider's actions until its activity is required. All values between 0 and 1 express different levels of "compliance"; users that have a certain probability to wait for the service provider's action or to take an uncontrollable action. Additionally, to allow for a wider range of possible user behaviour, besides antagonistic users, we model users that decide their actions randomly. Figure 7 outlines the implementation of the parameterized user class. We discretized compliance probability p with integers ranging from 0 to 100. In the main block of our simulation, shown in Figure 8, we generate a workflow object, `WorklfowDriver driver`, a company object for that workflow, `Company company` and several parameterized users, `List<User> users`. The user objects, which contain a `run()` method, start the simulation; their results are gathered in a map storing for each end state a triple over the total number of users in that state, the average number of steps and the average gas; further information about individual users is gathered in the background.

In our model, we differentiate users solely based on their compliance parameters. Remark that service providers may need to invest significant effort in determining their users' parameters to adjust their offers and fine-tune services. Discovering crucial user parameters is not trivial and requires extensive testing. Therefore, we investigate in the case study presented in Section 6 whether user compliance is a suitable way to capture realistic user behavior. Adjusting the compliance allows us to investigate different game settings without having to collect additional new data.

6 Case Study

6.1 Context

GrepS[2] is a company offering programming skill evaluations for Java programmers. GrepS is commissioned by external companies for recruiting, training, and certification. The service that GrepS provides is based on prior research [6]. *Customers* of GrepS are typically companies that hire or train developers, which are

[2] See the webpage of GrepS for further details: https://www.greps.com/.

```
class ParametricUser(
    WorkflowDriver driver, Company company, Int compliance)
  implements User
{
    Bool finished = False;
    String current_state = "start";

    Unit run() {
      while (!finished) {
          WorkflowTasks possible_tasks =
              await driver!available_tasks(current_state);
          WorkflowTasks u_tasks = uncontrollable_tasks(possible_tasks);
          WorkflowTasks c_tasks = controllable_tasks(possible_tasks);
          if (u_tasks != Nil && c_tasks != Nil)
          {
              // Choose with the given probability whether to perform
              // an uncontrollable or controllable action.
              if (random(100) < compliance) {
                  this.offerControllableAction();
              } else {
                  this.uncontrollableAction(u_tasks);
              }
          } else if (u_tasks != Nil) {
              // Only uncontrollable actions available
              this.uncontrollableAction(u_tasks);
          } else if (c_tasks != Nil) {
              // Only controllable actions available
              this.offerControllableAction();
          } else {
              // No action available: User reached an end state
              finished = True;
          }
      }
    }

    Unit offerControllableAction() {
      Maybe<WorkflowTask> action =
          await company!controlledAction(current_state);
      switch (action) {
        Just(the_task) => {
          current_state = target_state(the_task);
        }
        Nothing => finished = True;
      }
    }
}
```

Fig. 7: Implementation of the parameterized user class.

```
// Main block.
{
  // Parameters to set for the chosen experiment
  Int n_users = ...
  Int obedience = ...

  // Instantiate the underlying workflow model
  WorkflowDriver driver = new WorkflowDriver("non_det");
  // Create company object
  Company company = new Company(driver);

  // Create parameterized user objects
  List<User> users =
      await util!create_users(n_users, driver, company, obedience);

  // Aggregate results (end state => (count, avg. steps, avg. gas))
  Map<String, Triple<Int, Int, Float>> end_states =
      await util!collectUsersInMap(users, file);
}
```

Fig. 8: Creation of actors.

the *users* of the service. Users are normally given one to two weeks to complete their programming skill evaluation.

A typical programming skill evaluation requires the user to complete three phases using GrepS: (1) sign-up, (2) solve a set of authentic programming tasks, and (3) approve to share the results (via a skill report) with the customer, i.e. the commissioning company. In a *successful* user journey, all three phases are completed in order and the customer receives the report. In an *unsuccessful* journey, the user permanently stops using the service at any phase, or does not approve the sharing of the results with the customer.

The data we analyze are system logs with recorded events from the interactions between users and the GrepS system. These system logs are an extended version of the logs published as part of the work of Kobialka *et al.* in [32], as the logs we use also contain the programming skill evaluations that are calculated by the GrepS system. An extract of the extended data is shown in Figure 10. In this previous work, we report on the systematic generation and analysis of the GrepS user journey game. Figure 9 displays a simplified illustration of the task-solving and approval phase, leaving out the previously analyzed sign-up phase (states T0–T7). Controllable transitions are depicted as solid lines, uncontrollable transitions as dashed lines; transitions with positive weight are colored green, and those with negative weight are red. Each task during phase 2 consists of a pair of states: the first state is the solving of a task and the second is user feedback on the task. State T8 is a set-up task that is not used to evaluate skill, and T9 its corresponding user feedback. States T10–T17 are alternating tasks and feedback with T10 being the first practice task, T12 the second task, T14 the third task, and T16 the fourth task; the respective feedback is submitted after each single task. After each task has been submitted by the user, the system attempts to score the solution to the task and update the user's skill evaluation based on all solutions that have been scored so far. If the scoring process is successful, the log is updated ("Overall scores updated"). The increasing weights on edges along T8–T18 result from more users completing their tasks, i.e. users struggle with the first three tasks but from the third task on are all subsequent tasks completed [32]. In state T18, the user is informed that all tasks have been completed and explains the next steps that are to be completed within a specific number of workdays (as agreed with each GrepS customer in a service level agreement, SLA). States T21–T25 form the review phase, and T25 is the user approval for sharing the required report.

6.2 Evaluations of Users' Programming Skills by the GrepS System

The extended system logs capture many events with evaluated programming skills per user. We consider the last evaluation event as the *final* evaluated score (it captures the overall score of a user), and refer to the previous evaluation events as *tentative* scores. For each task solved by the user, the system evaluates the tentative skill level based on all available information (i.e., the current and any previous tasks that can be scored automatically). Note that the final score may involve partially human-graded tasks on dimensions such as readability, proper use of variable names, or other aspects that cannot be evaluated automatically. Thus, a user may have only one final skill score but can have many tentative skill scores during phase 2.

The unit of measurement used for the skill score is logits (i.e., the logarithm of the odds), which is frequently used within education or psychology to represent differences in skills and abilities on an interval scale using the Polytomous Rasch Model. A 5 on the scale used by GrepS is defined as the averagely skilled professional Java developer reported by Bergersen *et al.* [6], who also reported a standard deviation of skill scores of 1.3 logits. Note that this type of scale does not allow for ratio comparisons of skills (e.g., "someone is twice as

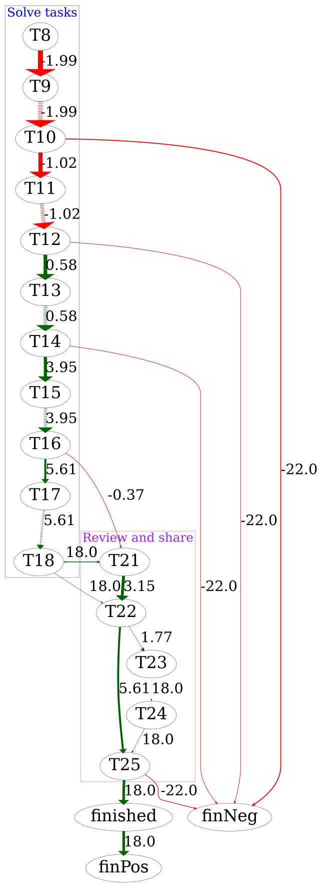


Fig. 9: GrepS user journey game (excerpt): task solving and approval phases.

Timestamp	···	***Metadata***
5245944	···	Registered
5780525	···	Registered
6104714	···	Activated
6104714	···	Logged in: Web page
6106191	···	Overall scores updated: [rasch.skill: 2.59 . . .

Fig. 10: Extract of GrepS' system logs.

skilled") because the number zero skill is not defined. However, the magnitude of differences is nevertheless constant across the range of the scale so that the magnitude of differences can be represented using a standardized effect size [17]. For example, a difference of 1.5 logits (i.e., a difference between 5.0 and 6.5, or 4.0 and 5.5) in skill would be considered a "large" effect by conventional standards (i.e., Cohen's $d = 0.8$).

Skill evaluations in the provided system logs reveal an association between successful and unsuccessful journeys. Figure 11 compares the box-plots of skill scores for successful journeys, in orange, and unsuccessful journeys, in blue. We ignore all journeys without a skill evaluation[3] and only use the final evaluated score per user. The current comparison contains a *survival* bias since we ignore the skill levels of all users that did not receive any skill evaluation. For the 11 unsuccessful journeys, the median skill level is 4.2, thus about 0.6 standard deviation for the less-than-average developer. For the 20 successful journeys, the median recorded skill level was about the same as an average developer (5.1). Both box-plots in Figure 11 range from 2 to 7. Observe that the data distribution of unsuccessful journeys has more variance: its lower quartile reaches significantly lower than the lower quartile of the successful journeys. The upper quartile of the successful journeys reaches higher and is denser than the one of unsuccessful journeys.

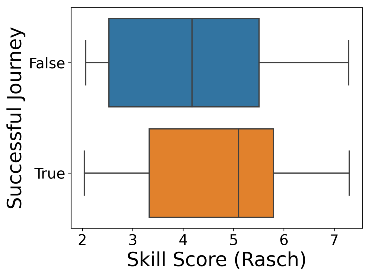


Fig. 11: Skill level comparison for successful and unsuccessful journeys from GrepS' system logs.

Both box-plots indicate that the GrepS service is currently better suited for at-least average proficient developers. Developers below average, with a score of less than 3.5, have a clear disadvantage. The log also demonstrates that above-average developers fail and below-average developers succeed. The outcome of the journey is not determined by the skill level but also by other factors.

[3] We observed that system logs contain a large amount of very short unsuccessful journeys with no events containing scores. Including all these journeys would negatively bias the comparison and therefore we remove them from the comparison.

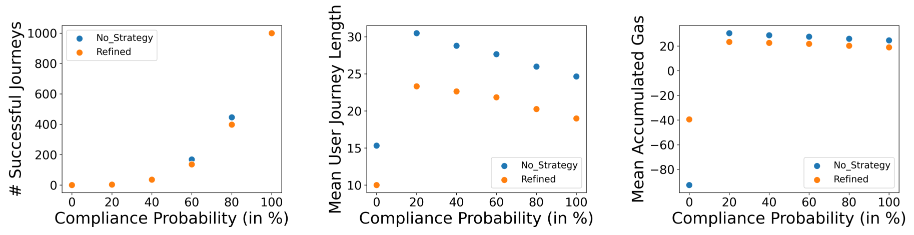
6.3 Simulation Analysis

We conduct several simulated scenarios in which we instantiate the parameterized user journey game, evaluate different service provider strategies, and test varying levels of user compliance, as a parametric behavior for users. We investigate the impact of different game strategies and the implications for the service provider to use the refined strategy to have a successful journey outcome and improve the user experience.

Building the baseline of the model. The initial model is constructed by modeling the GrepS user journey game and importing it into ABS, along with its corresponding UPPAAL game strategies for the service provider. We implement three strategies that the service provider can use: (1) a random one for a random selection between all available actions to the next transition in the game (no strategy), (2) a nondeterministic strategy, and (3) a refined strategy, minimizing the number of steps to reach a positive outcome, concretely, to reach the state `finPos`, see Figure 9. Strategies (2) and (3) are exported from UPPAAL STRATEGO into ABS, as described in Section 5.1. The users are randomized; i.e., they take a random, uncontrollable actions.

We first check that our ABS model reproduces the results generated in UPPAAL. The user is parameterized in its compliance, instantiated with a fixed probability at run-time, see Section 5.3. We calibrate our ABS model with suitable user compliance settings and sufficient many simulated users, such that simulation results are aligned with the results from UPPAAL STRATEGO. By doing so, our model reproduces the average amount of gas when reaching a final state and the average number of steps for the nondeterministic and refined strategy each.

Exploring alternative scenarios. We experimented with a less restrictive model by adapting the underlying game, where we removed some of the assumptions that were needed for the game analysis in UPPAAL STRATEGO. In particular, in UPPAAL STRATEGO the analysis requires a guaranteeing strategy for the service provider, thus, users are not allowed to give up in the middle of their journeys and those actions (solid lines, representing transitions in the game to the final negative state `finNeg`, see Figure 9) are controlled by the service provider. We adapt the model by making these interactions uncontrollable (therefore, controlled by users). Further, the UPPAAL STRATEGO game assumes that users always have precedence over the service provider. We additionally adapt our simulated users with a compliance parameter p, with probability $1 - p$ for each user to select a random, uncontrollable action, thereby lifting the assumption of adversary users. In the active object model, the generated strategies no longer guarantee a successful user journey outcome. However, the simulations still allow us to explore the GrepS user journey game, using one of the strategies. We observed that in the parameterized active object model with uncontrollable transitions to `finNeg`, the random strategy aligns with the nondeterministic strategy since there are no activities that the service provider is not allowed to select. Therefore, we only compare the random strategy with the refined strategy.



(a) Successful journeys (b) User journey length (c) Accumulated gas

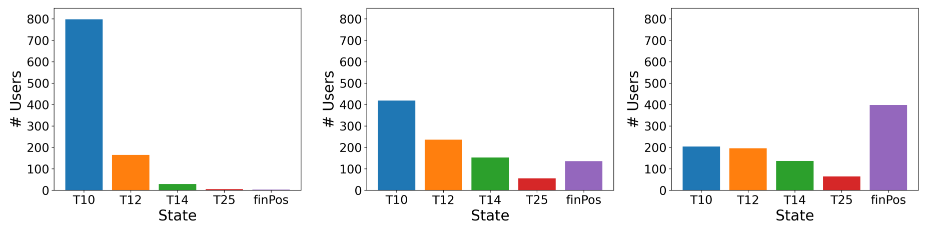
Fig. 12: Comparison of different compliance probabilities and strategies.

Figure 12 compares different aspects of the parameterized active object model with the two strategies, where all transitions leading to unsuccessful journey outcomes are user-controlled. The chance of taking such transitions (and therefore determining the outcome of the journey) is given by the compliance probabilities. Figure 12a displays the number of successful journeys for given compliance probabilities (the parameter p), we run simulations with a total of 1000 users, and with different probabilities for noncompliance or giving up $(1-p)$, ranging from 0% to 100% and increasing p with 20% in each simulation. When comparing the mean user journey length, the refined strategy improves the random strategy drastically since user journeys are significantly shorter in the refined strategy, see Figure 12b. Figure 12c compares the accumulated gas, revealing that the refined strategy reduces the accumulated gas for compliant users slightly. For noncompliant users with a short journey, the refined strategy improves the average accumulated gas from an average below -80 to -40.

Moreover, we investigate the different states where users give up their journey in the adapted ABS model, according to different compliance levels. Figure 13 shows the simulation results. With decreasing compliance levels, more users leave the journey, due to several states that allow users to give up, the number of users reaching the positive outcome shrinks rapidly, see Figures 13a and 13b. For compliant users, see Figure 13c, the service provider has good chances to guide the user to a successful outcome.

Skill level and compliance. Observations from the system logs show that the average GrepS user is closely comparable with the simulations that consider 80% compliance, see Figure 13c, and that $\frac{2}{3}$ of the users are successful. Concretely, in the provided log 18% of users gave up after the first task event, which aligns with the 20% decrease of users in the simulations, but only 9% gave up at the second and third tasks, which does not entirely align with the 16% and 12% decrease of users that is shown in the simulations, as well as 12% decrease of users at the reporting phase in the logs, with 6% decrease in the simulations.

We also investigate the relationship between the final skill score of users, detailed in Section 6.2, and the compliance parameter. Figure 14 shows the correspondence between the length of user journeys and the final skill level per



(a) 20% compliance (b) 60% compliance (c) 80% compliance

Fig. 13: Comparison of states where users stop in the simulated user journeys with different compliance probabilities; the company's goal is to maximize the reachability of finPos.

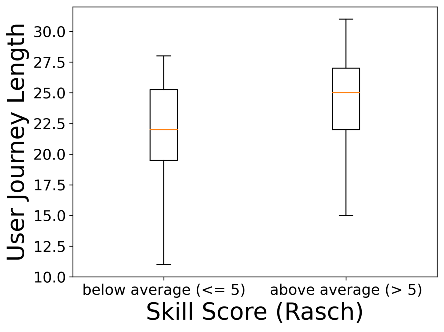
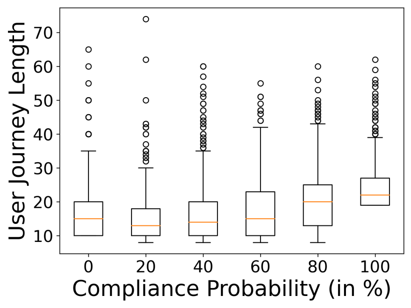


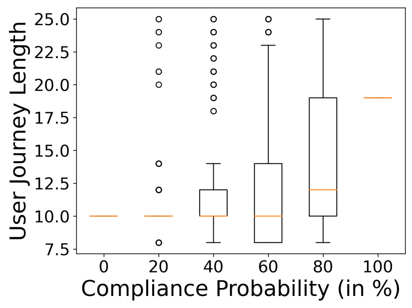
Fig. 14: Box-plots over user journey lengths per skill level in the system logs.

user. Results are grouped by skill level, where below or equal to 5 are sorted into the group of developers that are "below average", otherwise they are sorted into the group "above average". In comparison, Figure 15 displays the simulated box-plots over user journey lengths for different compliance levels for the random strategy (Figure 15a), and the refined strategy (Figure 15b). While the length of user journeys vary from the user journey lengths observed in the logs when using the random strategy in the simulations, the refined strategy produces user journeys with comparable lengths to those observed in the system logs. We further investigated the ratio of successful and unsuccessful journeys in the two skill groups. When ignoring users without a skill evaluation, 56% of unsuccessful user journeys belong to users that are scored "below average", and 73% of successful user journeys belong to users that are scored "above average". These values correspond to a compliance probability of about 85% for "below average" users and of more than 90% for "above average" users.

While not all aspects of the user-specific behavior could be replicated, our model is capable of differentiating different user groups as observed in the real-world system logs. By introducing one parameter for user compliance, we determine whether a user acts before the service provider and chooses an uncontrollable action. Whereas compliance appears to be a suitable notion to capture observed user behavior, users are in reality influenced by a wide range of parameters that are not independently recorded. We parameterized the modeling of user-specific behaviors and gained detailed insights into different kinds of users. The model adequately captured not only user journey lengths but also the distribution of final states and the number of successful journeys.



(a) Random strategy



(b) Refined strategy

Fig. 15: Box-plots over varying compliance levels and strategies in simulations.

6.4 Prescriptions

By using the simulation tool of the ABS active object language, we are able to adjust model details to conform to what is contained in the user logs. Previous game analysis required us to assume that users do not leave the journey, otherwise, no guaranteeing strategy could have been established. The active object model was adjusted to consider user parameters in the simulations, capturing various kinds of users. These adaptations allow service providers to evaluate the impact of possible changes on their services before implementing them. Several possible changes in either the strategy, the model, or both can be evaluated and the most promising ones can be implemented. Further, including user parameters in the model, allow us to adjust the simulation results towards the targeted users. In our case study, we compared different strategies and confirmed the suggestions from the model checker that the refined strategy is superior to the nondeterministic or random strategy. The strategies were generated in overapproximated games and tested in a more realistic setting. We could elaborate on differences between user groups and model them with proxy parameters, extracted from the system logs.

6.5 Evaluation

Our conclusions from the simulation analysis (Section 6.3) and new opportunities from exploring alternative scenarios (Section 6.4) can be summarised as follows:

1. more positive outcomes are related to “above average” skilled users,
2. compliance is a relevant proxy for user behavior (although with a still unresolved relation to programming skill), and
3. “what-if” scenarios may be used to simulate changes to the existing system for evaluating alternative directions for technical development in the future.

These insights have been further discussed with a long-term employee of the company, and third author of this paper, for their review. We summarize the feedback below.

Regarding point 1, more positive outcomes, GrepS is a user-focused service and is aware that different groups of users behave differently in their system. Depending on what a skill evaluation is used for, there are also situations where it is most sensible for a user to discontinue using the service. For example, if the first couple of programming tasks appear too difficult in a recruitment setting, a developer may opt out of the process and look for a different job. It is also known that less skilled developers are more resource-demanding in terms of needed support during the process, probably as a partial function of how well the user reads and understands the process and its requirements. At the same time, for a user-focused service, it is important to know which user groups are the key users, for which most of the resources should be used to keep satisfied with the service. Internally, the company is aware that less skilled users are less likely to complete all the programming tasks in Phase 2, or share an unsatisfying result during Phase 3.

Regarding point 2, compliance as a proxy of user behavior, companies need to challenge and further refine their own understandings of their key user groups. User parameters such as compliance—the willingness or capability to follow instructions—provide a more nuanced view than merely using programming skill evaluation to explain why some user journeys are unsuccessful. Compliant users tend to be more successful in their journey and have fewer problems solving the presented tasks, but it is at present unclear what the conceptual overlap is between "compliance" and factors such as technical skill or motivation.

Regarding point 3, the prescriptive analysis used to investigate "what-if" scenarios and to challenge assumptions that do not hold, GrepS is positive to evaluate such functionality more closely. For example, if a user in the present setup of the system stops solving a task, e.g. in states T10, T12, or T14, this user is unsuccessful. The simulation model could then attempt to answer the hypothetical question of what would happen if GrepS introduces a "user recovery" state where the sole goal is to bring the user back to the system, for example, by asking for feedback (is the user satisfied?), reminding the user (has the user forgot to continue?) or providing other kinds of targeted information (do the user know that valuable feedback is possible even though the report is not shared with GrepS' customer?). By estimating both expected costs (development time) and expected success (probabilities that users continue), such a simulation may yield better predictions of how many additional users would complete the analysis. If the simulation reveals that the additional users in the positive final state from a hypothetical intervention exceeds the cost of implementing it, such an intervention might be prioritized. Simulations of large and complex systems where relevant factors are parameterized, seem preferable to heavily relying on heuristics of what works (and doesn't) that require extensive experience to validate.

7 Conclusion and Future Work

This paper presents an active object simulation framework for user journeys. The framework can be combined with strategy analysis for service providers, based on model checking user journey games. We considered strategies generated in the model checker Uppaal Stratego and showed that the results of model checking can be reproduced in our framework. Then, we extended the framework to parameterize the users' compliance with the intended user journey, and estimated how resilient different service provider strategies are to non-cooperating users. The active object framework allows prescriptive analysis, where the impact of changes can be evaluated before implementing them in the real system.

Previous analysis based on user journey games, using Uppaal Stratego, over-approximates the service provider behavior, to establish strategies that guarantee a successful outcome. The active object simulation framework alleviates these assumptions, making the user journey model more realistic. The simulation framework uses two measures for the modeled user journeys: the total number of actions taken in the journey and the accumulated cost. When adapting the user journey game to the simulation framework, one has to evaluate if the strategy generated from the user journey game is compatible with the active object model. Otherwise, a random strategy might outperform a refined strategy. Therefore, it is important to compare refined strategies to a valid baseline, i.e. a random or nondeterministic strategy, and, if necessary, update the refined strategy to the new assumptions.

We present an industrial case study from GrepS, a small company offering programming skill evaluations to other companies. We investigated users with “below average” and “above average” proficiency. Our simulations reproduced findings from the Greps log, suggesting that GrepS is configured for “above average” proficient users. These users have a higher chance for a successful user journey with shorter user journeys than “below average” proficient users. In the case study, the active object model harmonized well with the refined strategy, user journey lengths were reduced and the final gas was kept at comparable levels in the system logs and simulations.

The presented active object simulation framework opens many interesting possibilities for future work. One obvious extension is to make the active object framework resource-sensitive, exploiting the resource-model of ABS [29]. The current model only considers gas as a resource, but every interaction between service provider and user has a duration and also requires physical resources, e.g. interactions with a GrepS employee. A time- and resource-sensitive model allows scenarios to be explored that show response times under various loads and “what-if” scenarios; e.g., whether adding personnel to answer user messages in a certain state of the user journey would increase overall completion rates. Such extension could consider load balancers that distribute or delegate activities in the service to workers with limited resources, mimicking resource management in cloud-based distributed systems, as previously modeled and analyzed using ABS with time and resources [28, 35, 37, 48].

The current model is Markovian, as the next decision only depends on the current state. However, the model does provide access to the accumulated gas of users (i.e., the sum of the weights from previous interactions with the service provider). This allows richer models of decision-making to be investigated, where the current decision not only depends on the users' compliance parameter but also on past experiences by taking into account the accumulated gas, capturing how much "steam" the user has left to continue the journey. Accordingly, it would also be interesting to investigate further model parameters and their influence on successful user journeys (e.g., to fine-tune compliance or to capture other user behavior characteristics).

Conflict of Interest The third author has financial interests in the company (GrepS) that owns the skill testing tool evaluated in the case study in this work.

References

1. W. M. P. van der Aalst: Process Mining - Data Science in Action. Springer, 2 edn. (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Albert, E., de Boer, F.S., Hähnle, R., Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L., Wong, P.Y.H.: Formal modeling and analysis of resource management for cloud architectures: an industrial case study using Real-Time ABS. Serv. Oriented Comput. Appl. **8**(4), 323–339 (2014). https://doi.org/10.1007/s11761-013-0148-0
3. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) Proc. 19th Intl. Conf. on Computer Aided Verification (CAV 2007). Lecture Notes in Computer Science, vol. 4590, pp. 121–125. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_14
5. Berendes, C.I., Bartelheimer, C., Betzing, J.H., Beverungen, D.: Data-driven customer journey mapping in local high streets: A domain-specific modeling language. In: Pries-Heje, J., Ram, S., Rosemann, M. (eds.) Proc. Intl. Conf. on Information Systems - Bridging the Internet of People, Data, and Things (ICIS 2018). Association for Information Systems (2018), https://aisel.aisnet.org/icis2018/modeling/Presentations/4
6. Bergersen, G.R., Sjøberg, D.I.K., Dybå, T.: Construction and validation of an instrument for measuring programming skill. IEEE Trans. Software Eng. **40**(12), 1163–1184 (2014). https://doi.org/10.1109/TSE.2014.2348997
7. Bernard, G., Andritsos, P.: CJM-ex: Goal-oriented exploration of customer journey maps using event logs and data analytics. In: Clarisó, R., Leopold, H., Mendling, J., van der Aalst, W.M.P., Kumar, A., Pentland, B.T., Weske, M. (eds.) Proc. BPM Demo Track and BPM Dissertation Award co-located with 15th Intl. Conf. on Business Process Modeling (BPM 2017). CEUR Workshop Proceedings, vol. 1920. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1920/BPM_2017_paper_172.pdf
8. Bernard, G., Andritsos, P.: A process mining based model for customer journey mapping. In: Franch, X., Ralyté, J., Matulevicius, R., Salinesi, C., Wieringa, R.J. (eds.) Proc. Forum and Doctoral Consortium Papers at the 29th Intl. Conf. on

Advanced Information Systems Engineering (CAiSE 2017). CEUR Workshop Proceedings, vol. 1848, pp. 49–56. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1848/CAiSE2017_Forum_Paper7.pdf

9. Bernard, G., Andritsos, P.: CJM-ab: Abstracting customer journey maps using process mining. In: Mendling, J., Mouratidis, H. (eds.) Information Systems in the Big Data Era - Proc. CAiSE Forum 2018. Lecture Notes in Business Information Processing, vol. 317, pp. 49–56. Springer (2018). https://doi.org/10.1007/978-3-319-92901-9_5
10. Bernard, G., Andritsos, P.: Contextual and behavioral customer journey discovery using a genetic approach. In: Welzer, T., Eder, J., Podgorelec, V., Latific, A.K. (eds.) Proc. 23rd European Conference on Advances in Databases and Information Systems (ADBIS 2019). Lecture Notes in Computer Science, vol. 11695, pp. 251–266. Springer (2019). https://doi.org/10.1007/978-3-030-28730-6_16
11. Bertolini, C., Liu, Z., Srba, J.: Verification of timed healthcare workflows using component timed-arc Petri nets. In: Proc. Second International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2012). pp. 19–36. Springer (2013). https://doi.org/10.1007/978-3-642-39088-3_2
12. Bezirgiannis, N., de Boer, F.S., Johnsen, E.B., Pun, K.I., Tapia Tarifa, S.L.: Implementing SOS with active objects: A case study of a multicore memory system. In: Hähnle, R., van der Aalst, W.M.P. (eds.) Proc. 22nd International Conference on Fundamental Approaches to Software Engineering (FASE 2019). Lecture Notes in Computer Science, vol. 11424, pp. 332–350. Springer (2019). https://doi.org/10.1007/978-3-030-16722-6_20
13. Bitner, M.J., Ostrom, A.L., Morgan, F.N.: Service blueprinting: A practical technique for service innovation. California Management Review **50**(3), 66–94 (Apr 2008). https://doi.org/10.2307/41166446
14. de Boer, F., Serbanescu, V., Hähnle, R., Henrio, L., Rochas, J., Din, C.C., Johnsen, E.B., Sirjani, M., Khamespanah, E., Fernandez-Reyes, K., Yang, A.M.: A Survey of Active Object Languages. ACM Comput. Surv. **50**(5), 76:1–76:39 (Oct 2017). https://doi.org/10.1145/3122848
15. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Optimal strategies in priced timed game automata. In: Lodaya, K., Mahajan, M. (eds.) Proc. 24th Intl. Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2004). Lecture Notes in Computer Science, vol. 3328, pp. 148–160. Springer (2004). https://doi.org/10.1007/978-3-540-30538-5_13
16. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: A model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013). Lecture Notes in Computer Science, vol. 7795, pp. 185–191. Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_13
17. Chinn, S.: A simple method for converting an odds ratio to effect size for use in meta-analysis. Statistics in medicine **19**(22), 3127–3131 (2000)
18. Crosier, A., Handford, A.: Customer journey mapping as an advocacy tool for disabled people: A case study. Social Marketing Quarterly **18**(1), 67–76 (Mar 2012). https://doi.org/10.1177/1524500411435483
19. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc Petri nets. In: Proc. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012). Lecture Notes in Com-

puter Science, vol. 7214, pp. 492–497. Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_36

20. David, A., Jensen, P.G., Larsen, K.G., Legay, A., Lime, D., Sørensen, M.G., Taankvist, J.H.: On time with minimal expected cost! In: Cassez, F., Raskin, J. (eds.) Proc. 12th International Symposium on Automated Technology for Verification and Analysis (ATVA 2014). Lecture Notes in Computer Science, vol. 8837, pp. 129–145. Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_10
21. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) Proc. 21st Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015). Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16
22. Følstad, A., Kvale, K.: Customer journeys: A systematic literature review. Journal of Service Theory and Practice **28**(2), 196–227 (Mar 2018). https://doi.org/10.1108/JSTP-11-2014-0261
23. Fornell, C., Mithas, S., Morgeson, F.V., Krishnan, M.: Customer satisfaction and stock prices: High returns, low risk. Journal of Marketing **70**(1), 3–14 (Jan 2006). https://doi.org/10.1509/jmkg.70.1.003.qxd
24. Halvorsrud, R., Kvale, K., Følstad, A.: Improving service quality through customer journey analysis. Journal of Service Theory and Practice **26**(6), 840–867 (Nov 2016). https://doi.org/10.1108/JSTP-05-2015-0111
25. Halvorsrud, R., Mannhardt, F., Johnsen, E.B., Tapia Tarifa, S.L.: Smart journey mining for improved service quality. In: Carminati, B., Chang, C.K., Daminai, E., Deng, S., Tan, W., Wang, Z., Ward, R., Zhang, J. (eds.) Proc. IEEE International Conference on Services Computing (SCC 2021). pp. 367–369. IEEE (2021). https://doi.org/10.1109/SCC53864.2021.00051
26. Halvorsrud, R., Sanchez, O.R., Boletsis, C., Skjuve, M.: Involving users in the development of a modeling language for customer journeys. Software and Systems Modeling pp. 1–30 (2023). https://doi.org/10.1007/s10270-023-01081-w
27. Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A core language for abstract behavioral specification. In: Proc. 9th International Symposium on Formal Methods for Components and Objects (FMCO 2010). Lecture Notes in Computer Science, vol. 6957, pp. 142–164. Springer (2010). https://doi.org/10.1007/978-3-642-25271-6_8
28. Johnsen, E.B., Pun, K.I., Tapia Tarifa, S.L.: A formal model of cloud-deployed software and its application to workflow processing. In: 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM). pp. 1–6 (Sep 2017). https://doi.org/10.23919/SOFTCOM.2017.8115501
29. Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L.: Integrating deployment architectures and resource consumption in timed object-oriented models. Journal of Logical and Algebraic Methods in Programming **84**(1), 67–91 (Jan 2015). https://doi.org/10.1016/j.jlamp.2014.07.001
30. Kamburjan, E., Hähnle, R., Schön, S.: Formal modeling and analysis of railway operations with active objects. Sci. Comput. Program. **166**, 167–193 (2018). https://doi.org/10.1016/j.scico.2018.07.001
31. Kobialka, P., Mannhardt, F., Tapia Tarifa, S.L., Johnsen, E.B.: Building user journey games from multi-party event logs. In: Montali, M., Senderovich, A., Weidlich, M. (eds.) Process Mining Workshops (ICPM 2022). Lecture Notes in Business Information Processing, vol. 468, pp. 71–83. Springer (2022). https://doi.org/10.1007/978-3-031-27815-0_6, Proc. 3rd Intl. Workshop on Event Data and Behavioral Analytics (EdbA 2022)

32. Kobialka, P., Tapia Tarifa, S.L., Bergersen, G.R., Johnsen, E.B.: Weighted games for user journeys. In: Proc. 20th International Conference Software Engineering and Formal Methods (SEFM 2022). Lecture Notes in Computer Science, vol. 13550, pp. 253–270. Springer (2022), https://doi.org/10.1007/978-3-031-17108-6_16
33. Lammel, B., Korkut, S., Hinkelmann, K.: Customer experience modelling and analysis framework a semantic lifting approach for analyzing customer experience. In: Proc. 6th Intl. Conf. on Innovation and Entrepreneurship (IE 2016). GSTF (Dec 2016). https://doi.org/10.5176/2251-2039_IE16.10
34. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997). https://doi.org/10.1007/s100090050010
35. Lin, J., Lee, M., Yu, I.C., Johnsen, E.B.: Modeling and simulation of spark streaming. In: Barolli, L., Takizawa, M., Enokido, T., Ogiela, M.R., Ogiela, L., Javaid, N. (eds.) Proc. 32nd IEEE Intl. Conf. on Advanced Information Networking and Applications (AINA 2018). pp. 407–413. IEEE Computer Society (2018)
36. Lin, J., Mauro, J., Røst, T.B., Yu, I.C.: A model-based scalability optimization methodology for cloud applications. In: Proc. 7th International Symposium on Cloud and Service Computing (SC^2 2017). pp. 163–170. IEEE Computer Society (2017). https://doi.org/10.1109/SC2.2017.32
37. Lin, J., Yu, I.C., Johnsen, E.B., Lee, M.: ABS-YARN: A formal framework for modeling hadoop YARN clusters. In: Stevens, P., Wasowski, A. (eds.) Proc. 19th International Conference on Fundamental Approaches to Software Engineering (FASE 2016). Lecture Notes in Computer Science, vol. 9633, pp. 49–65. Springer (2016). https://doi.org/10.1007/978-3-662-49665-7_4
38. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS 95). Lecture Notes in Computer Science, vol. 900, pp. 229–242. Springer (1995). https://doi.org/10.1007/3-540-59042-0_76
39. Razo-Zapata, I.S., Chew, E.K., Proper, E.: VIVA: A visual language to design value co-creation. In: Proc. 20th Conference on Business Informatics (CBI 2018). vol. 01, pp. 20–29. IEEE (Jul 2018). https://doi.org/10.1109/CBI.2018.00012
40. Rosenbaum, M.S., Otalora, M.L., Ramírez, G.C.: How to create a realistic customer journey map. Business Horizons **60**(1), 143–150 (2017). https://doi.org/10.1016/j.bushor.2016.09.010
41. Schlatte, R., Johnsen, E.B., Kamburjan, E., Tapia Tarifa, S.L.: Modeling and analyzing resource-sensitive actors: A tutorial introduction. In: Damiani, F., Dardha, O. (eds.) Coordination Models and Languages. Lecture Notes in Computer Science, vol. 12717, pp. 3–19. Springer (2021). https://doi.org/10.1007/978-3-030-78142-2_1
42. Schlatte, R., Johnsen, E.B., Kamburjan, E., Tapia Tarifa, S.L.: The ABS simulator toolchain. Sci. Comput. Program. **223**, 102861 (2022). https://doi.org/10.1016/j.scico.2022.102861
43. Schlatte, R., Johnsen, E.B., Mauro, J., Tapia Tarifa, S.L., Yu, I.C.: Release the beasts: When formal methods meet real world data. In: It's All About Coordination - Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab. Lecture Notes in Computer Science, vol. 10865, pp. 107–121. Springer (2018). https://doi.org/10.1007/978-3-319-90089-6_8
44. Terragni, A., Hassani, M.: Analyzing customer journey with process mining: From discovery to recommendations. In: Proc. 6th International Conference on Future Internet of Things and Cloud (FiCloud 2018). pp. 224–229. IEEE (Aug 2018). https://doi.org/10.1109/FiCloud.2018.00040

45. Terragni, A., Hassani, M.: Optimizing customer journey using process mining and sequence-aware recommendation. In: Proc. 34th Symposium on Applied Computing (SAC 2019). pp. 57–65. ACM Press (Apr 2019). https://doi.org/10.1145/3297280.3297288
46. Tueanrat, Y., Papagiannidis, S., Alamanos, E.: Going on a journey: A review of the customer journey literature. Journal of Business Research **125**, 336–353 (Mar 2021). https://doi.org/10.1016/j.jbusres.2020.12.028
47. Turin, G., Borgarelli, A., Donetti, S., Damiani, F., Johnsen, E.B., Tapia Tarifa, S.L.: Predicting resource consumption of kubernetes container systems using resource models. Journal of Systems and Software (2023). https://doi.org/10.1016/j.jss.2023.111750, to appear
48. Turin, G., Borgarelli, A., Donetti, S., Damiani, F., Johnsen, E.B., Tapia Tarifa, S.L.: Predicting resource consumption of kubernetes container systems using resource models. Journal of Systems & Software **203**, 111750 (Sep 2023). https://doi.org/10.1016/j.jss.2023.111750
49. Vandermerwe, S., Rada, J.: Servitization of business: Adding value by adding services. European Management Journal **6**(4), 314–324 (Dec 1988). https://doi.org/10.1016/0263-2373(88)90033-3
50. Wong, P.Y.H., Albert, E., Muschevici, R., Proença, J., Schäfer, J., Schlatte, R.: The ABS tool suite: modelling, executing and analysing distributed adaptable object-oriented systems. Int. J. Softw. Tools Technol. Transf. **14**(5), 567–588 (2012). https://doi.org/10.1007/s10009-012-0250-1