Weighted Games for User Journeys *

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

[1] University of Oslo, Oslo, Norway
{p.kobialka,sltarifa,gunnab,einarj}@ifi.uio.no
[2] GrepS B.V., Utrecht, the Netherlands
gunnar@greps.com

Abstract. The servitization of business is moving industry to business models driven by customer demand. Customer satisfaction is connected with financial rewards, forcing companies to investigate in their users' experience. User journeys describe how users manoeuvre through a service. Today, user journeys are typically modelled graphically, and lack formalization and analysis support. This paper proposes to formalize user journeys as weighted games between the user and the service provider. We further propose a data-driven construction of such games, derived from system logs using process mining techniques. As user journeys may contain cycles, we bound the number of iterations in each cycle and develop an algorithm to unfold user journeys into acyclic weighted games. These can be model checked using Uppaal Stratego to uncover potential challenges in how a company interacts with its users and to derive company strategies to guide users better in their journeys. Our analysis pipeline was evaluated on an industrial case study; it revealed design challenges within the studied service and could be used to derive suitable recommendations for improvement.

Keywords: User journeys · Data-driven model construction · Games · Model checking · UPPAAL.

1 Introduction

The *servitisation of business* [37], a concept of creating added value to products by offering services, is a major practice embraced by most (if not all) successful companies. Such companies are interested in the analysis of their services, which until now has mostly focused on the *managerial* perspective, where the service is analysed with respect to the companies' view. Recent tendencies are shifting the focus from the company's view to the end-users view, where a positive experience and impression that a user has while engaging in the service, has shown to have a positive impact on the financial reward of a company [17]. Thus, companies aim to analyse and improve their services, based on their users' satisfaction.

* This work is part of the *Smart Journey Mining* project, funded by the Research Council of Norway (grant no. 312198).

User journeys (also called customer journeys) analyse services from the user perspective [30]: A user journey is inherently a goal-oriented process, because humans engage in a service with a goal in mind. The user moves through the journey by engaging in so-called touchpoints, which are either actions performed by the user or a communication event between the user and a service provider. We here assume that users only engage in one touchpoint of a service at a time.

Tools are currently lacking the analysis of user journeys [21], which hinders their operational use. User journey diagrams are usually generated by hand, and the user perspective is derived from interviews with experts and users, e.g. [20,30]. This process has been highly successful, discovering points of failure in the studied services and, as a result, providing advice to companies on how to improve their services. However, this manual process is best suited for relatively small services and a restricted number of users. For services with thousands of users, journey diagrams need to be automatically generated and analysed.

This paper proposes a formalization of user journeys as *weighted games* [12] between users and a service provider, and a method to derive these games from process logs. Our aim is to use these games to analyse services and to suggest service improvements such that service providers always have a strategy to guide their users towards a desired goal. We capture the user perspective of services by means of so-called *gas*. The term is inspired by blockchain technology such as Ethereum, where gas refers to the cost necessary to perform a transaction on the network. In our work, the gas quantitatively reflects how moves in the user journey contribute to the users reaching their goal. Consequently, the moves in the derived games are weighted and accumulated into the gas of the journeys, which allows to compare and analyse journeys using model checkers such as Uppaal Stratego [15] or PRISM-games [13], and to give strategic recommendations to service providers. In short, our contributions are: (1) a formalization of user journeys as weighted games; (2) a pipeline to automatically derive and model check weighted games; and (3) an industrial case study that evaluates the feasibility of our approach.

Related Work: We discuss related work with respect to the modelling of user journeys and the use of data-driven techniques to discover user journeys. We are not aware of prior work that uses automatic verification methods to check properties for user journeys.

User journeys aim to improve service design by describing how users interact with services [16, 36]. Modelling notations for user journeys aim to support the so-called blueprinting [11], i.e., to create an anticipated model of a service. There are various notations to create diagrams for user journeys [5, 14, 19, 24, 29, 30]; these diagrams are mostly handmade but some digital support exists; for example, a semantic lifting into ontologies has been used to visualize fixed aspects of a model [24]: the data sent, the communication channels and devices used, etc. Berendes *et al.* propose in [5] the *high street journey modelling language* (HSJML) tailored to journeys in shopping streets. Razo-Zapata *et al.* propose the *VIVA* modelling language with focus on interactions [29]. In contrast, our work aims to use data-driven techniques [2] to automatically discover user jour-

ney diagrams and formal methods to automatically check properties of user journeys and derive recommendations for improving the service under analysis.

The *Customer Journey Modelling Language* (CJML) [18, 20] captures the end-users point of view. CJML distinguishes planned and actual user journeys, which represent the journey as planned as part of the service design and as perceived by the user, respectively. Our work is part of a project [21] on tool support for data-driven user journey modelling in CJML. Whereas previous work on CJML manually quantifies user experience collected through user feedback questionnaires, our work aims to capture the journeys as perceived by the user in a data-driven manner, based on system logs.

Data-driven techniques for process discovery allow us to discover user journeys. Harbich *et al.* [22] use mixtures of Markov models to derive user journey maps. Bernard *et al.* [8, 10] study process mining [2] for user journeys, such as hierarchical clustering to explore large numbers of journeys [7] and process discovery techniques to generate user journey maps at different levels of granularity [9]. Terragni and Hassani [33] apply process mining to user journey web logs to build process models, and improve the results by clustering journeys. This work has been integrated with a recommender system to suggest service actions that maximize key performance indicators [34], e.g., how often the product page is visited. In our work we propose a data-driven method to discover models of user centric journeys with multiple actors, building on on existing techniques.

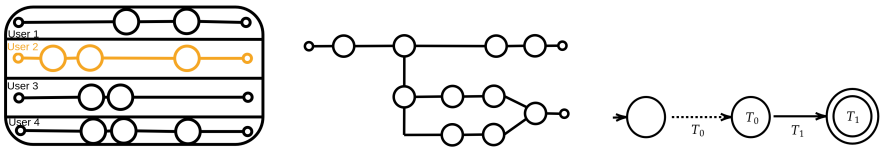
Outline: Section 2 introduces foundational definitions needed for weighted games and the model checking suite Uppaal that we use for analysis. The formal model for user journeys is introduced in Sects. 3–5 and model checked in Sect. 6. Section 7 discusses the implementation, Sect. 8 evaluates our approach in terms of an industrial case study and Sect. 9 concludes the paper.

2 Preliminaries

We briefly summarise the formal notations and tools that we build on for the proposed user journey pipeline to analyse a service.

A *transition system* [28] is a tuple $S = \langle \Gamma, A, E, s_0, T \rangle$ with a set Γ of states, a set A of actions (or labels), a transition relation $E \subseteq \Gamma \times A \times \Gamma$, an initial state $s_0 \in \Gamma$ and a set $T \subseteq \Gamma$ of final states. A *weighted transition system* [35] $\mathcal{S} = \langle S, w \rangle$ extends the transition system S with a weight function $w : E \to \mathbb{R}$ that assigns weights to transitions.

Weighted games [12] are obtained from weighted transition systems by partitioning the actions A into *controllable* actions A_c, and *uncontrollable* actions A_u, where only actions in A_c can be controlled by the analyser, while actions in A_u are nondeterministically decided by an adversarial environment. When analysing games, we look for a *strategy* that guarantees a desired outcome, i.e. winning the game by reaching a certain state. The strategy is given by a partial function $\Gamma \to Act_c \cup \{\lambda\}$ that decides on the action of the controller in a given state (here, λ denotes the "wait" action, letting the adversary move).



Log Directly Follows Graph Game

Fig. 1: Creation of the Journey Model.

Uppaal Tiga [4] can be used to analyse reachability and safety properties for games expressed using (timed) transition systems, extending the model checker Uppaal [25]. Uppaal Tiga checks whether there is a strategy under which the behaviour satisfies a control objective, denoted `control: P` for a property P. Property P is expressed in computational tree logic [3], an extension of propositional logic that is used to express properties along paths in a transition system. Recall that computational tree logic *state properties* ϕ can be decided in a single state; while *reachability properties* `E <>` ϕ express that the formula ϕ is satisfiable in some reachable state in a transition system; *safety properties* `E []` ϕ express that the formula ϕ is always satisfied in all the states of *some path* in a transition system and `A []` ϕ expresses that ϕ is always satisfied in all the states of *all paths* of a transition system. Similarly, *liveness properties* `A <>` ϕ express that the formula ϕ will eventually be satisfied in all the paths in a transition system and the formula ϕ `-->` ψ expresses that satisfying formula ϕ leads to satisfying formula ψ.

Uppaal Stratego [15] can be used to analyse and refine a strategy generated by Uppaal Tiga with respect to a quantitative attribute like weights. Uppaal Stratego is a statistical model checker [27]; it extends Uppaal for stochastic priced timed games and combines simulations with hypothesis testing until statistical evidence can be deduced.

3 From User Logs to Games

To capture the *user perspective* in games that model user journeys, user actions (representing communication initiated by the user) can be seen as controllable and the service provider's actions as uncontrollable. However, from an analytical perspective, it is more interesting to treat user actions as uncontrollable and the service provider's actions as controllable. The service provider is expected to have suitable reactions for all possible user interactions. Ideally, the service provider should not rely on the user to make the journey pleasant. By treating user actions as uncontrollable, we can expose the worst behaviour of the service provider, and thereby strengthen the user-centric perspective promoted by journey diagrams. Games for user journeys are then defined as follows:

Definition 1 (User journey games). *A* user journey game *is a weighted game* $\langle \Gamma, A_c, A_u, E, s_0, T, T_s, w \rangle$*, where*

- Γ *are states that represent the touchpoints of the user journey,*

- A_c *and* A_u *are disjoint sets of actions respectively initiated by the service provider and the user,*
- $E \subseteq \Gamma \times A_c \cup A_u \times \Gamma$ *are the possible actions at the different touchpoints,*
- $s_0 \in \Gamma$ *is an initial state,*
- $T \subseteq \Gamma$ *are the final states of the game,*
- $T_s \subseteq T$ *are the final states in which the game is successful, and*
- $w : E \to \mathbb{R}$ *specifies the weight associated with the different transitions.*

The process of deriving such user journey games from user logs is illustrated in Fig. 1. In *Step 1*, we go from logs to a user journey model, expressed as a directly follows graph (DFG), and in *Step 2*, the DFG is extended to a game. The derivation of weights for the transitions is discussed in Sect. 4.

Step 1. We use a *directly follows graph* (DFG) as an underlying process model to capture the order of events in an event log; a DFG is well-suited as the process model provided that users only engage in one touchpoint at a time. DFGs are derived from event logs by means of *process discovery* [2]. An *event log* L is a multi-set of journeys. A journey $J = \langle a_0, \ldots, a_n \rangle$ is a finite and ordered sequence of events a_i from a universe $\mathscr{A}$. We construct the DFG of an event log L as a transition system $S = \langle \Gamma, A, E, s_0, T \rangle$ where the states Γ capture the event universe, $\Gamma \subseteq \mathscr{A} \cup \{s_0\} \cup T$. Every sequence of events is altered to start in the start state s_0 and to end in a dedicated final state $t \in T$. The set of actions A is the union of the event universe and the final states, $A = \mathscr{A} \cup T$. The transition relation E includes a triple (a_i, a_{i+1}, a_{i+1}) if a_i is directly followed by a_{i+1} in some $J \in L$; we can traverse from state a_i to state a_{i+1} by performing the action a_{i+1}. Here reaching a state in S is interpreted as the corresponding event in L has already been performed. By construction, the DFG S obtained from log L can replay every observed journey in L. However S may capture more journeys than those present in L, since for example S may contain transitions with loops.

Step 2. The DFG is now transformed into a game. Observe that the DFG captures the temporal ordering of events but it does not directly differentiate the messages sent by the user to the service provider from those sent by the service provider to the user. For simplicity, let us assume that this information is either part of the events in the logs or known in advance from domain knowledge concerning the event universe. The mined DFG can then be extended into a game by annotating the actions that are *(un)controllable*.

4 Capturing User Feedback in User Journey Games

We now extend the games derived from system logs into weighted games by defining a *gas* function reflecting user feedback. We develop a gas function that is automatically calculated and applied to the transitions of the game, depending on the traversal and entropy that is present in the corresponding event log. Informally, the gas function captures how much "steam" the consumer has left to continue the journey. Less steam means that the user is more likely to abort

the journey and more steam means that the user is more likely to complete the journey successfully. Assuming that the service provider attempts to provide the best possible service, its goal is to maximize gas in a journey. The adversarial user aims for the weaknesses in the journey and therefore minimizes the gas. Formally, the weight function $w : E \to \mathbb{R}$ maps the transitions E of a game to weights, represented as reals. Given a log L and its corresponding game, we compute the weight for every transition $e \in E$.

Since user journeys are inherently goal-oriented, we distinguish successful and unsuccessful journeys; the journeys that reach the goal are called *successful* and the remaining journeys are considered to be *unsuccessful*. This is captured by a function majority : $E \times L \to \{-1, 1\}$ that maps every transition $e \in E$ to $\{-1, 1\}$, depending on whether the action in the transition appears in the majority of journeys in L that are unsuccessful or successful, respectively. Ties arbitrarily return -1 or 1.

Many actions might be part of both successful and unsuccessful journeys. For this reason, we use Shannon's notion of *entropy* [32]. Intuitively, if an action is always present in unsuccessful journeys and never in successful ones, there is certainty in this transition. The entropy is low, since we understand the context in which this transition occurs. In contrast, actions involved in both successful and unsuccessful journeys have high entropy. The entropy is calculated using

1. the number of occurrences of an event in the transitions of successful journeys within the event log L, denoted $\#_L^{pos} e$, and the number of transitions in unsuccessful ones, denoted $\#_L^{neg} e$; and
2. the total number of occurrences of the event in L, denoted $\#_L e$.

The entropy H of transition e given the event log L is now defined as

$$H(e, L) = -\frac{\#_L^{pos} e}{\#_L e} \cdot log_2(\frac{\#_L^{pos} e}{\#_L e}) - \frac{\#_L^{neg} e}{\#_L e} \cdot log_2(\frac{\#_L^{neg} e}{\#_L e}) \ .$$

The weight function w that computes the weights of the transitions can now be defined in terms of the entropy function, inspired by decision tree learning [31]. Given an event log L, the weight of a transition e is given by

$$w(e) = ((1 - H(e, L)) \cdot \text{majority}(e, L) - C) \cdot M \ .$$

The constant C represents an aversion bias and is learned from the training set. It is used to model a basic aversion against continuous interactions. The sign of a transition depends on its majority. If the transition is mostly traversed on successful journeys, it is positive. Otherwise, it is negative. The inverse entropy factor quantifies the uncertainty of transitions. The constant M scales the energy weight to integer sizes (our implementation currently requires integer values, see Sect. 7).

The gas of a journey quantitatively reflects the history of that journey, allowing us to not only compare the weights of transitions but also to compare (partial) journeys. The gas $\mathcal{G}$ of a journey $J = \langle a_0, \ldots, a_n \rangle$ with transitions

Algorithm 1 k-bounded loop unrolling

Input: Weighted Game $S = \langle \Gamma, A_c, A_u, E, s_0, T, T_s, w \rangle$, constant $k \in \mathbb{N}^+$
Output: Acyclic Weighted Game $S' = \langle \Gamma', A_c, A_u, E', s_0, T', T_s, w' \rangle$

Initialize $S' = \langle \emptyset, A'_c, A'_u, \emptyset, s_0, T', T_s, w \rangle$ and queue $Q = [s_0]$
$C \leftarrow \{c \mid c \text{ is simple cycle in } S\}$
while not EMPTY(Q) **do**
    state $s \leftarrow$ FIRST(Q)
    for $t \in \{t \mid (s,t) \in E\}$ **do**
        hist $\leftarrow$ PUSH(HISTORY(s), t)
        allSmaller $\leftarrow$ **True**
        canTraverse $\leftarrow$ **False**
        if REPETITIONS($c, hist$) $\geq k$ for all cycle $c \in C$ **then**
            allSmaller $\leftarrow$ **False**
        if !allSmaller **then**
            $P \leftarrow$ ALLSIMPLEPATHS(S, t, T)
            for path $p \in P$ **do** ▷ check whether cycle might be partially traversed
                $hist' \leftarrow$ MERGE($hist, p$)
                if REPETITIONS($c, hist'$) $\leq k$ for all cycle $c \in C$ **then**
                    canTraverse $\leftarrow$ **True** ▷ cycle can be partially traversed
        if allSmaller $\vee$ canTraverse **then**
            state t' copy of t with history $hist$
            PUSH(Q, t')
            ADDTRANSITION($(s, t'), S'$) ▷ Copies weight to w' and actor to A'_c, A'_u
return S'

$e_0, \ldots e_{n-1}$ is defined as the sum of the weights along the traversed transitions:

$$\mathcal{G}(J) \coloneqq \sum_{i=0}^{n-1} w(e_i) \ .$$

5 Finite Unrolling of Games

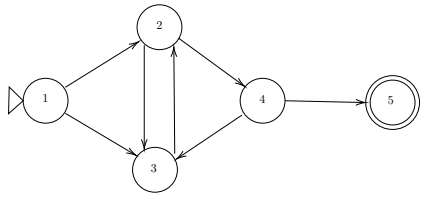
The generated weighted games may contain loops, which capture unrealistic journeys (since no user endures indefinitely in a service) and hinder model checking. Therefore, the weighted games with loops are transformed into acyclic weighted games using a breadth-first search loop unrolling strategy bounded in the number of iterations per loop. The transformation is implemented in an algorithm that preserves the original decision structure and adds no additional final states.

The algorithm for *k-bounded loop unrolling* (shown in Algorithm 1) returns an acyclic weighted game, where each loop is traversed at most k times. The unrolling algorithm utilizes a breadth-first search from the initial state s_0 in combination with a loop counting to build an acyclic weighted game. In the algorithm, the state s denotes the current state that is being traversed. To traverse the paths in the weighted game, we use a queue Q to store the states that need to be traversed, a set C containing all the cycles in the graph (where each cycle

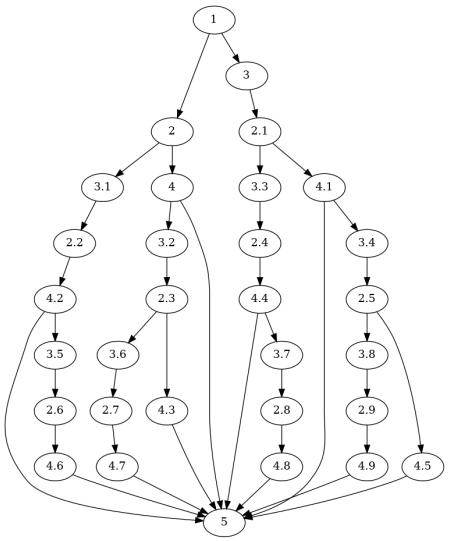
is a sequence of states), and the function allSimplePaths(S, s, T) that returns all paths in the weighted game S from s to any final state $t \in T$. The extended graph is stored in the acyclic game S'. A state in a cycle can be traversed if it has been visited less than k times (see Lines 9–10). The function repetitions checks the number of traversals. If the counter for one cycle is k, the algorithm checks whether the cycle can be partially traversed (see Lines 11–16).

Partial traversals guarantee that we reach a final state without closing another loop. The partial traversal does not increase the count of another cycle to $k+1$ (Lines 14–16). Every state stores its history (a sequence of visited states), which can be retrieved using the function history. Line 14 increases the current history by including a (partial) path through the loop. This check iterates through all paths from the current state to any final state. If state t can be traversed, it is added to the acyclic game (Lines 17–20). A copy t' of t is added to the queue Q, the transition (s, t'), its weight and actor are added to S' using the function addTransition. The resulting weighted game can be reduced. All states outside a cycle can be merged into the same state. This can either be done after unrolling the whole game or on the fly while unrolling.

Example. Figure 2 illustrates the unrolling algorithm (for simplicity, we ignore transition weights and do not distinguish controllable and uncontrollable actions in the example). Starting from the cyclic weighted game in Fig. 2a, the algorithm with $k = 1$ generates the acyclic weighted game in Fig. 2b. The input contains two loops: $C = \{[2, 3], [2, 4, 3]\}$. Starting at state 1, we can traverse two neighbour states which both are part of the cycles. Thus, both transitions are inserted to S', and Q is updated to $[2, 3]$. Continuing with state 2, all reachable transitions are again inserted as the corresponding cycles have not been fully traversed. Names of copies of the states that are already present once in the graph are incremented (the first occurrence of state 3 is called 3, the second 3.1, the third, 3.2, etc.) The algorithm continues until the first loop $2, 3, 2$ is closed. In this case, it is not possible to traverse again to state 3 without closing the loop $[2, 3]$. Only state 4 and its corresponding loop can be traversed (see



(a) Cyclic Game



(b) Acyclic Game

Fig. 2: Unrolling Example.

Fig. 2b, left branch). As result of the state reduction, all final states are merged into one (removing the copies originally introduced by the algorithm).

Properties. The unrolling algorithm preserves the decision structure of the initial weighted game. By construction, acyclic weighted games do not traverse cycles in the initial game $k+1$ times. Loops can be traversed partially to ensure that every final state in the acyclic weighted game is also a final state in the initial weighted game. Only unreachable states are excluded in the acyclic game. No further final states or "dead ends" are introduced. The algorithm also preserves the local decisions between controllable and uncontrollable actions, so the strategies found on the unrolled weighted game carry over to the initial weighted game.

6 Model Checking User Journeys

In this section we describe how to model check properties for user journeys and generate strategies to improve user journeys, using acyclic weighted games. The analysis of a weighted game gives formal insights into the performance of a service. We introduce generic properties that capture the user's point of view on a user journey. The analysis in this paper uses the STRATEGO extension for UPPAAL [15], which supports non-deterministic priced games and stochastic model checking. STRATEGO allows to model check reachability properties within a finite number of steps, when following a strategy (therefore the need for acyclic games). STRATEGO constructs a strategy that satisfies a property P, so that the controller can not be defeated by the non-deterministic environment. We detail some strategies and properties of interest for games derived from user journeys.

Guiding users to a target state. A company needs a suitable plan of (controllable) actions for all possible (uncontrollable) actions of their users when guiding them through a service. In UPPAAL STRATEGO we define the following strategy:

```
strategy goPos = control: A<> Journey.finPos .
```

Model checking this property returns true if and only if there exists a company-strategy `goPos` such that the positive target state `finPos`, indicating that the journey is successful, is eventually reached in all paths. The corresponding strategy (given as a pseudo-code) can be produced with the UPPAAL TIGA command-line tool VERIFYTGA. If the verification fails, the company should be advised to simplify their service and offer more support to avoid unsuccessful user journeys.

Analysing user feedback. We can use the gas function and a liveness property to analyse the desired accumulated feedback at the end of successful user journeys:

```
Journey.finPos --> gas > 0 under goPos .
```

This property checks that in general users have balancing experiences within their journeys, when the company follows the `goPos` strategy.

We can also check the feedback levels along the journey. The following property checks that a user never falls below a defined constant feedback `C`:

```
control: A[] gas > C under goPos .
```

Fluctuations in the feedback level of users can be revealed using simulations. Uppaal uses an implicit model for the passage of time to guarantee termination of statistical queries and simulations, using an upper time bound `T`, as specified in [15]. The following query simulates `X` runs through the system using the `goPos` strategy, where each run has `T` as a time bound:

```
simulate X [t<=T]{Journey.finPos, gas} under goPos .
```

The time bound is set to a value that guarantees all runs to reach a final state.

Analysing the trajectory of user journeys. Reaching a final state in a journey with a positive feedback does not ensure a satisfying journey. The user might still visit every pitfall along the way. To provide a pleasant journey, a company is among others interested in minimising the expected number of steps. A strategy minimising the number of steps is refined as follows:

```
strategy goPosFast = minE(steps) [t<=T] :
    <> Journey.finPos under goPos .
```

This strategy can additionally be used to examine the expected lower bound of gas within a journey and the expected maximum value of accumulated gas at the end of a journey (denoted by `finalGas`):

```
E[t<=T; X] (min: gas) under goPosFast ,
E[t<=T; X] (max: finalGas) under goPosFast .
```

These values are computed with a time bound of `T` and over `X` runs.

7 Implementing the Pipeline to Analyse User Journeys

This section describes the implementation of the analysis pipeline detailed in Sects. 3–6. We focus on the implementation decisions made along the pipeline to facilitate the analysis. The pipeline is implemented in Python. The input to the pipeline are user logs of a service provided by a company and the output is a Uppaal model, which can be model checked by either the proposed properties in Sect. 6 or by other custom made properties using Uppaal Stratego. A source repository for our work on user journey games is available online [1].

We first mine the DFG from user logs and then remove transitions that were rarely traversed, to simplify the graph and make it robust. Leemans *et al.* describe two ways to build a robust DFG [26]: One can (1) remove either transitions from the graph or (2) remove journeys from the log and rebuild the graph. We used the first approach with a traversal threshold of three in this paper, since removing journeys requires larger datasets. This modification ensures

that the model only contains relevant journeys. We then enrich the graph with knowledge indicating which actions are controllable and uncontrollable. Since companies want to understand why on-boarded users reach their goal or quit in the middle of a journey, we decided to add to the model final states representing a positive endpoint, `finPos`, and a negative one, `finNeg`, respectively.

We generate a weighted transition system by computing a weight for each transition, as discussed in Sect. 4. The factor M scales the weights to integer sizes, required by UPPAAL's model checker. However, given that we simplify the DFGs, the log contains journeys that are not re-playable in the graph. Computing the gas of such journeys corresponds to the *alignment* problem [23, 26]. The alignment procedure consists of either allowing additional steps in the log without counterparts in the model or allowing steps in the model without steps in the log. Since the simplification of DFGs omits steps in the model, it was here sufficient to use the information given in the trace, without inferring further model steps. Optimal alignments can also be used to compute the gas.

As a final step, we unroll the weighted game with cycles, as described in Sect. 5, to obtain an acyclic weighted game, which is the output of the transformation and the input to UPPAAL for further analysis. Bounded constraints in the properties are introduced to the unrolled model to ensure termination. The analysis described in Sect. 6 is implemented and evaluated.

8 Evaluating the Analysis Pipeline

In this section we evaluate the implemented pipeline described in Sect. 7 in an industrial case study from the company GrepS. The full details of the case study are given in the accompanying artefact.[1]

8.1 Context

GrepS is a company that provides analysis and measurement of programming skills for the Java programming language. The service is research based [6]. Typical customers are organisations hiring or training software developers. The *users* of the service are developers who receive a request from a customer organization to complete a skill analysis within a specific time frame, typically 1–2 weeks.

The service consists of a sign-up phase followed by a phase where the user solves programming tasks in an authentic programming environment, which includes an instructional task and a practice task. Finally, the service analyses the user's skills and requests the user to share the skill report with the customer.

The customer pays GrepS for each skill report it receives. In a *successful* use of the service, a user successfully completes three phases: (1) sign up, (2) solve all programming tasks, and (3) review and share the skill report with the customer. In an *unsuccessful* use of the service, the user permanently stops using the service or does not want to share the skill report with GrepS' customer.

[1] An artefact for the implementation and evaluation of the analysis pipeline in this paper is available: `https://doi.org/10.5281/zenodo.6962413`.

`strategy goPos = control: A<> Journey.finPos`	True
`Journey.finPos --> e > 0 under goPos`	False
`control: A[] gas > -41 under goPos`	True
`E[t<=100; 500] (max: steps) under goPos`	28.5
`E[t<=100; 500] (min: gas) under goPos`	-26.7
`E[t<=100; 500] (max: finalGas) under goPos`	60
`strategy goPosFast = minE(steps) [t<=100] : <> Journey.finPos under goPos`	True
`E[t<=100; 500] (max: steps) under goPosFast`	20.9
`E[t<=100; 500] (min: gas) under goPosFast`	-20.1
`E[t<=100; 500] (max: finalGas) under goPosFast`	35

Fig. 4: Analysis of the weighed game generated from the user logs of GrepS.

Anonymised user logs were provided by GrepS in the form of tabular data. The logs contain events with various fields; only the fields *Timestamp*, that gives the order of events, and *Metadata*, containing meta-information on the kind of event, were used to generate the weighted game. An extract of the data is shown in Fig. 3. For our purposes, only the order of the events was of interest, as given by the *Timestamp*.

Timestamp	···	***Metadata***
5245944	···	Registered
5780525	···	Registered
6104714	···	Activated
6104714	···	Logged in: Web page
⋮	⋮	⋮

Fig. 3: Extract of GrepS' user logs.

The validation of the analysis pipeline includes observations of the weighted game and the model checking of the properties as outlined in Sect. 6, the results are summarised in Fig. 4. The analysis results were used to provide recommendations for GrepS to improve their service. These recommendations were validated by the third author, a long-term employee of the company who has experience in handling feedback from both users and customers.

8.2 Observations in the Weighted Game

The generated cyclic user journey game, which still contains loops, is shown with events (or touchpoints) T and weighted transitions in Fig. 5. In the figure, the transition thickness indicates how often a transition was traversed and dashed lines represent uncontrollable transitions. Positive (negative) transitions are green (respectively, red). Transitions traversed three times or less were removed from the graph.

The derived weights already allow us to make some interesting observations. The weighted game shows negative weights (about -1 to -2) through Phase 1 (T0–T5), up until the practice task has been completed (T12) in Phase 2 (T6–T20). After that, the weights are positive (about $+1$ to $+5$) and increase steadily for each new task. Phase 3 (T21–T26) also has positive weights through the user journey; here, a developer logs back into the web system after having

completed all tasks (T19), waits for the report to be ready (T21), and finally approves the sharing of the report with GrepS' customer (T26).

Phase 1 shows two negative weights for some users that involve more touchpoints than what the planned journey entails: (1) T4 captures an error where a virtual computer does not spin up correctly thereby requiring the user to contact support; (2) there are a cyclical negative weights between T6–T8 where a user starts receiving instructions for Phase 2, but stops and then returns to the system again at a later time. Phase 3 also has negative weights due to deviations from the planned journey, for example when the user does not login after the report is available (T24).

The figure also shows a strong negative weight (of -22) when a user does not submit the practice task in T11, resulting in a negative outcome, a transition to `finNeg`. Seen from a user perspective, Fig. 6 shows the four touchpoints where most users stop using the service: 18% of all users quit after finishing the practice task (T10), which is twice that of users who stop after the first (T12, 9%) and second task (T14, 9%); 12% of the users do not want to share their report (T25). The blue line shows how many users remain using the service in percent after each of the four touchpoints.

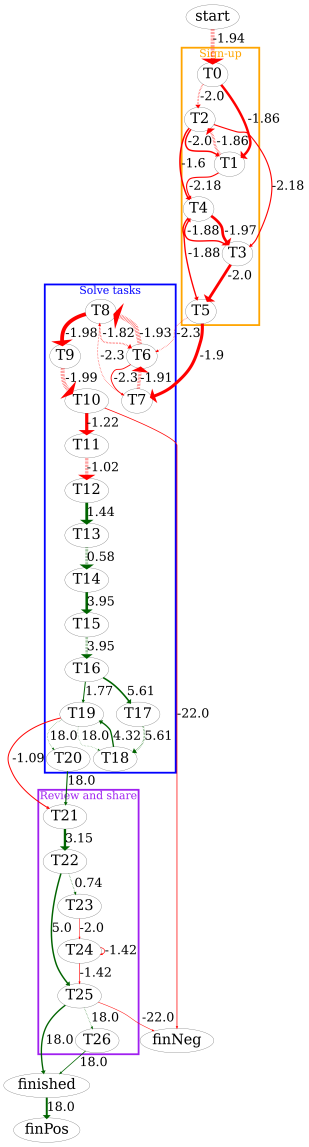


Fig. 5: The weighted game using GrepS' event logs.

8.3 Model Checking the Case Study

The accumulated feedback along the paths of the journey supports the observations on unsuccessful journeys (Sect. 8.2). Figure 7 shows 10 simulations with the `goPos` and `goPosFast` strategies; the lines show the amount of gas (accumulated feedback) along the journey. We here used $k = 1$ for the unrolling. For all simulations, the gas has an initial dip with a steep increase afterwards. The results in Fig. 4 support the observations in Sect. 8.2. Observe that the `goPos` strategy cannot prevent the gas from falling below 0; in fact, it can fall as low as -41 along the journey with an expected minimum of -26.7.

Depending on the application context, multiple factors can contribute to an optimised journey. The strategy `goPosFast` was introduced in Sect. 6 as a refinement of `goPos`. It searches for an optimal strategy towards a successful final state, while minimising the expected number of steps. The lower part of Fig. 4

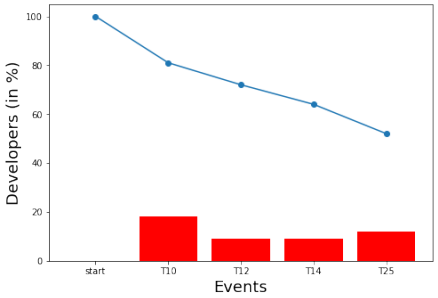


Fig. 6: Events in unsuccessful journeys.

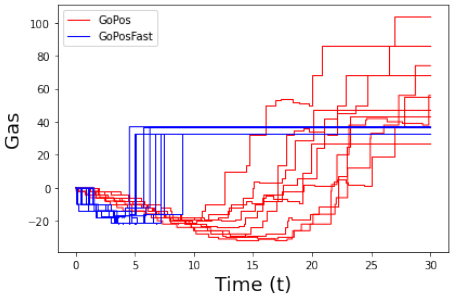


Fig. 7: UPPAAL simulations.

evaluates the queries under `goPosFast`. The simulations of the refined strategy, in Fig. 7, shows a smaller dip than with the `goPos` strategy. It improves the expected minimum feedback by 6.6 units and reduces the expected length of the journey by seven steps. The expected maximum final feedback is also reduced from 60 to 35.

8.4 Recommendations from the Observations and Analysis

From the company's perspective, several key takeaways have been identified from the weighted game, the simulations, and the model checking of properties:

- The instructional task and practice tasks during Phase 2 should be integrated into a single task that is more motivating for the user to complete.
- Users that disconnect from the service for several days after having progressed to the instructional, practice, or first task should be prompted to continue by, e.g., automatically sending a motivational email.
- The sign-up process should be simplified if possible.

The weighted game detects challenges early in Phase 2; in fact this is reassuring for our analysis, as prior work at GrepS has reported that the users struggle more during the first three tasks [6]. However, a question that arises from our analysis of the derived user journey game is whether good user support during deviations from the planned journey may result in better overall satisfaction than if the planned journey had no deviations. It seems plausible that unplanned journeys that involve technical problems result in less motivated users who are less likely to successfully complete the journey. However, interactions with support may also result in additional service to the user that may result in positive weights in the overall game.

In summary, the case study demonstrates that the analysis of games derived from system logs can be used to discover weaknesses in designed user journeys, and to improve and optimise these journeys. For the concrete case study, the company needs to implement additional actions in their service, which will improve user satisfaction and reduce costs in terms of resources.

9 Conclusions and Future Work

This paper proposes a novel analysis pipeline to gain insights into user journeys. We presented a method for the data-driven generation of formal models to analyse user journeys, using weighted games. To the best of our knowledge, this is the first automatic analysis pipeline using formal methods in the context of service science and user journeys. The paper proposes a method to automatically analyse derived models and thereby gain insights into the user journeys in a service, where all decisions and recommendations can be reasoned and explained. The model is not subject to human inference but is generically built from user logs.

The derived model preserves a user-centric point of view. We mine a directly follows graph from user logs, and extend this graph to a game by considering the actions of the user as uncontrollable and those of the service provider as controllable. Weights are introduced to the game by a gas function which maps transitions in the game to numerical values (in the real domain). Cycles in the derived graph are unrolled to generate an acyclic weighted game. The unrolling algorithm preserves weights, actions and final states from the initial graph. The resulting acyclic weighted game can be analysed with respect to properties expressed as Uppaal Stratego queries using the Uppaal model checker.

The proposed analysis pipeline was evaluated on an industrial case study and revealed challenges to the planned user journey of the service provider. The analysis of the derived game demonstrated that users' experiences fall in their accumulated feedback during the initial phases of the service. Our recommendations were reviewed and approved by an expert on user feedback in the company.

The work presented here opens many interesting possibilities for further work. Our work so far has assumed that users and service providers have perfect knowledge of each other's possible actions, such that the analysis could be done with the Stratego extension for Uppaal [15]. Generally, knowledge about planned user journeys varies between services and between users. We plan to explore imperfect information games, where, e.g., knowledge about user actions is not completely known. In this setting, the analysis could be based on probabilistic priced games, using the model checker PRISM-games [13].

Furthermore, the current analysis is restricted to strategies for unrolled models, which give insights from a k-bounded loop unrolling but does not generalise for unseen values $> k$. We would like to generate strategies for the initial model and not only for the unrolled model. We plan to integrate our work with existing modelling languages for user journeys in the service science domain, such as CJML [18, 20], to automate the analysis of user journey models that are manually reviewed today, and to provide feedback from our analysis in the visual language of these models. We are currently investigating the scalability of the proposed method on system logs for user journeys that are significantly larger than the case study considered here.

References

1. User Journey Games Repository. `https://github.com/smartjourneymining/User-Journey-Games`
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer (2016).
3. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Proc. 19th Intl. Conf. on Computer Aided Verification (CAV 2007). LNCS 4590, pp. 121–125. Springer (2007)
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: Proc. Intl. Conf. on Information Systems (ICIS 2018). Association for Information Systems (2018)
6. Bergersen, G.R., Sjoberg, D.I., Dyba, T.: Construction and Validation of an Instrument for Measuring Programming Skill. IEEE Transactions on Software Engineering **40**(12), 1163–1184 (Dec 2014)
7. Bernard, G., Andritsos, P.: CJM-ex: Goal-oriented exploration of customer journey maps using event logs and data analytics. In: 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)
8. Bernard, G., Andritsos, P.: A process mining based model for customer journey mapping. In: 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),
9. Bernard, G., Andritsos, P.: CJM-ab: Abstracting customer journey maps using process mining. In: Information Systems in the Big Data Era - Proc. CAiSE Forum 2018. Lecture Notes in Business Information Processing, vol. 317, pp. 49–56. Springer (2018)
10. Bernard, G., Andritsos, P.: Contextual and behavioral customer journey discovery using a genetic approach. In: Proc. 23rd Eur. Conf. on Advances in Databases and Information Systems (ADBIS 2019). LNCS 11695, pp. 251–266. Springer (2019)
11. Bitner, M.J., Ostrom, A.L., Morgan, F.N.: Service blueprinting: A practical technique for service innovation. California Management Review **50**(3), 66–94 (2008)
12. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Optimal strategies in priced timed game automata. In: Proc. 24th Intl. Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2004). LNCS 3328, pp. 148–160. Springer (2004)
13. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: A Model Checker for Stochastic Multi-Player Games. In: Proc. TACAS 2013. pp. 185–191. LNCS 7795, Springer (2013)
14. Crosier, A., Handford, A.: Customer Journey Mapping as an Advocacy Tool for Disabled People: A Case Study. Social Marketing Quarterly **18**(1), 67–76 (2012)
15. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Proc. TACAS 2015. pp. 206–211. LNCS 9035, Springer (2015)
16. Følstad, A., Kvale, K.: Customer journeys: A systematic literature review. Journal of Service Theory and Practice **28**(2), 196–227 (Mar 2018)
17. 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)
18. Halvorsrud, R., Boletsis, C., Garcia-Ceja, E.: Designing a modeling language for customer journeys: Lessons learned from user involvement. In: Proc. 24th Intl.

Conf. on Model Driven Engineering Languages and Systems (MODELS 2021). pp. 239–249. IEEE (2021)
19. Halvorsrud, R., Haugstveit, I.M., Pultier, A.: Evaluation of a modelling language for customer journeys. In: Proc. Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2016). pp. 40–48. IEEE Computer Society (2016)
20. 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)
21. Halvorsrud, R., Mannhardt, F., Johnsen, E.B., Tapia Tarifa, S.L.: Smart journey mining for improved service quality. In: Proc. IEEE Intl. Conf. on Services Computing (SCC 2021). pp. 367–369. IEEE (2021)
22. Harbich, M., Bernard, G., Berkes, P., Garbinato, B., Andritsos, P.: Discovering customer journey maps using a mixture of markov models. In: Proc. 7th Intl. Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2017). CEUR Workshop Proceedings, vol. 2016, pp. 3–7. CEUR-WS.org (2017),
23. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Trace alignment in process mining: Opportunities for process diagnostics. In: Proc. 8th Intl. Conf. on Business Process Management (BPM 2010). LNCS 6336, pp. 227–242. Springer (2010)
24. 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 (2016)
25. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997)
26. Leemans, S.J.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: Intl. Conf. on Process Mining (ICPM 2019). pp. 25–32. IEEE (2019)
27. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Proc. First Intl. Conf. on Runtime Verification (RV 2010). LNCS 6418, pp. 122–135. Springer (2010)
28. Plotkin, G.D.: A structural approach to operational semantics. J. Log. Algebraic Methods Program. **60-61**, 17–139 (2004)
29. Razo-Zapata, I.S., Chew, E.K., Proper, E.: VIVA: A visual language to design value co-creation. In: 20th Conf. on Business Informatics (CBI), pp. 20–29 IEEE (2018).
30. Rosenbaum, M.S., Otalora, M.L., Ramírez, G.C.: How to create a realistic customer journey map. Business Horizons **60**(1), 143–150 (2017)
31. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson (2020)
32. Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3), 379–423 (Jul 1948).
33. Terragni, A., Hassani, M.: Analyzing customer journey with process mining: From discovery to recommendations. In: Proc. 6th Intl. Conf. on Future Internet of Things and Cloud (FiCloud 2018). pp. 224–229. IEEE (Aug 2018)
34. 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)
35. Thrane, C., Fahrenberg, U., Larsen, K.G.: Quantitative analysis of weighted transition systems. Journal of Logic and Algebraic Programming **79**(7), 689–703 (2010)
36. Tueanrat, Y., Papagiannidis, S., Alamanos, E.: Going on a journey: A review of the customer journey literature. Journal of Business Research **125**, 336–353 (2021)
37. Vandermerwe, S., Rada, J.: Servitization of business: Adding value by adding services. Eur. Management Journal **6**(4), 314–324 (Dec 1988)