Temporal Analytics for Software Usage Models

Oana Andrei(✉) and Muffy Calder

School of Computing Science, University of Glasgow, Glasgow G12 8RZ, UK
oana.andrei@glasgow.ac.uk

Abstract. We address the problem of analysing how users *actually* interact with software. Users are heterogeneous: they adopt different usage patterns and each individual user may move between different patterns, from one interaction session to another, or even during an interaction session. For analysis, we require new techniques to model and analyse temporal data sets of logged interactions with the purpose of discovering, interpreting, and communicating meaningful patterns of usage. We define new probabilistic models whose parameters are inferred from logged time series data of user-software interactions. We formulate hypotheses about software usage together with the developers, encode them in probabilistic temporal logic, and analyse the models according to the probabilistic properties. We illustrate by application to logged data from a deployed mobile application software used by thousands of users.

1 Introduction

Software users are heterogeneous: they adopt different usage patterns for the same software, furthermore, each individual user may move between different patterns, from one interaction session to another, or even during a session. This may be for a variety of contextual reasons, for example, time of day, time since last session, length of session, purpose of engagement, environment (e.g. on a train or at a desk) or device (PC, tablet, wearable). To analyse these differing and dynamic usage patterns, we require new techniques to model and analyse temporal data sets of logged interactions with the purpose of discovering, evaluating, and communicating meaningful patterns of usage. We define new probabilistic models of software usage that are generated from logged data using statistical methods. Analysis is performed by model checking probabilistic temporal logic properties.

Our approach consists of:

- two new parametrised, admixture discrete-time Markov models that include latent (unobserved) and observed states and depend on a finite number K of usage patterns,
- segmented logged time series data encapsulating different usage time intervals, e.g. interactions over first day, first week, first month, second month,
- use of maximum-likelihood parameter estimation techniques for inferring model parameters,

A. Cerone and M. Roveri (Eds.): SEFM 2017 Workshops, LNCS 10729, pp. 9–24, 2018.
https://doi.org/10.1007/978-3-319-74781-1_1

– an encoding of the inferred usage models in the PRISM model checker [1] and use of temporal logics PCTL and PCTL* for analysis.

It is important to note we are not inferring the underlying software structure, which is determined by the designed functionality of the software. We are investigating the differing generating processes of software *usage* over a dynamic, heterogeneous population of a users. The logged data are user-initiated events: we refer to the time series data as *user traces*, where each user trace consists of all the logged sessions for that user/device. We identify basic common traits of usage emerging within such a population of users and classify users as *admixtures* of basic traits. In statistical modelling, in a mixture model each user trace is mapped to one single behavioural trait, whereas in admixture models each user trace has a distribution over all behavioural traits. To the best of our knowledge, inferring admixture temporal structures has not been described in prior work outside our group. Figure 1 summarises our approach. This work builds on two previous studies: an initial analysis of AppTracker using PAM models [2], and earlier, in [3], analysis of individual user models for a mobile game application. Major differences here include formal definitions of PAM, new GPAM model, generic properties, and use of parameter K as exploratory tool.

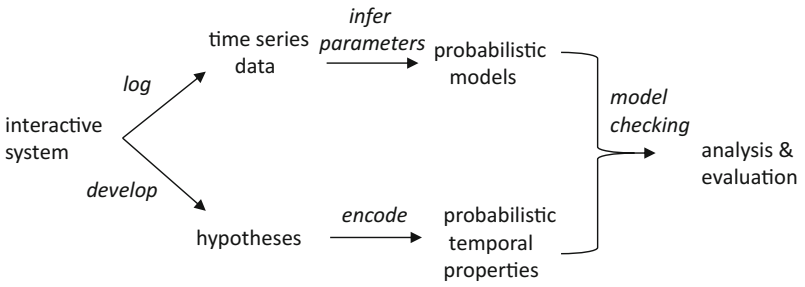


Fig. 1. Role of inference and probabilistic model checking in defining and analysing software usage models.

The main contributions of the paper are:

– two new latent state Markov models: population admixture model (PAM) and generalised population admixture model (GPAM),
– flattened form of the GPAM model as discrete time Markov chain and encoding in the probabilistic model checker PRISM,
– generic and GPAM-specific temporal logic properties,
– an example based on a deployed user-intensive mobile application software (mobile app) used by tens of thousands of users, working in close collaboration with the app developers.

In the next section we give background definitions. The new latent variable Markov models, PAM and GPAM, are introduced in Sect. 3. In Sect. 4 we define

classes of probabilistic temporal properties for analysing PAMs and GPAMs. In Sect. 5 we give an overview of our example interactive software and evaluate its usage by PAM and GPAM analysis. We reflect on our approach in Sect. 6 and conclusions are in Sect. 7.

2 Technical Background

2.1 Markov Models

We assume familiarity with Markov models, probabilistic logics PCTL and PCTL* with rewards, and model checking [1,4,5]; basic definitions are below.

Let $\mathcal{A}$ be a finite set of atomic propositions. A **discrete-time Markov chain (DTMC)**, also called a first-order Markov chain model, is a tuple $(S, \bar{s}, \mathbf{P}, \mathcal{L})$ where: S is the set of states; $\bar{s} \in S$ is the initial state, $\mathbf{P} : S \times S \to [0,1]$ is the transition probability function (or matrix) such that for all states $s \in S$ we have $\sum_{s' \in S} \mathbf{P}(s,s') = 1$; $\mathcal{L} : S \to 2^{\mathcal{A}}$ is a labelling function associating to each state s in S a set of valid atomic propositions from the set $\mathcal{A}$. A *path* (or execution) of a DTMC is a non-empty sequence $s_0 s_1 s_2 \ldots$ where $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. For DTMC $\mathcal{D}$ the set of paths starting from state s is $Path^{\mathcal{D}}(s)$. A transition is also called a *time-step*.

We will distinguish between latent (unobserved or hidden), and observed states. The standard latent variable Markov model is the hidden Markov model, which does not include transitions between observed states. Formally, a **first-order hidden Markov model (HMM)** is a tuple $(\mathcal{X}, \mathcal{Y}, \pi, A, B)$ where: $\mathcal{X}$ is the set of hidden (or latent) states $\mathcal{X} = \{1, \ldots, K\}$; $\mathcal{Y}$ is the set of observed states generated by hidden states; $\pi : \mathcal{X} \to [0,1]$ is an initial distribution, where $\sum_{x \in \mathcal{X}} \pi(x) = 1$; $A : \mathcal{X} \times \mathcal{X} \to [0,1]$ is the transition probability matrix, such that for all $x \in \mathcal{X}$ we have $\sum_{x' \in \mathcal{X}} A(x,x') = 1$; $B : \mathcal{X} \times \mathcal{Y} \to [0,1]$ is the observation probability matrix, such that for all $x \in \mathcal{X}$ we have $\sum_{y \in \mathcal{Y}} B(x,y) = 1$.

A less common, but more useful model than HMM for us, is the auto-regressive hidden Markov model, which also permits transitions between observed states. Formally, a **first-order auto-regressive hidden Markov model (AR-HMM)** is a tuple $(\mathcal{X}, \mathcal{Y}, \pi, A, B)$ where: $\mathcal{X}, \mathcal{Y}, \pi, A$ are as defined for HMM; $B : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [0,1]$ is the observation probability matrix, such that for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ we have $\sum_{y' \in \mathcal{Y}} B(x,y,y') = 1$.

2.2 Probabilistic Logics and Model Checking

Probabilistic Computation Tree Logic (PCTL) and its extension *PCTL** allow one to express a probability measure of the satisfaction of a temporal property by a DTMC. Their syntax is the following:

$$
\begin{array}{rl}
\textit{State formulae} & \Phi ::= \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathsf{P}_{\bowtie p}[\Psi] \mid \mathsf{S}_{\bowtie p}[\Phi] \\
\textit{PCTL Path formulae} & \Psi ::= \mathsf{X}\Phi \mid \Phi \,\mathsf{U}^{\leq N}\, \Phi \\
\textit{PCTL* Path formulae} & \Psi ::= \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \mathsf{X}\Psi \mid \Psi \,\mathsf{U}^{\leq N}\, \Psi
\end{array}
$$

where a ranges over a set of atomic propositions $\mathcal{A}$, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0,1]$, and $N \in \mathbb{N} \cup \{\infty\}$.

This is a minimal set of operators, the propositional operators *false*, disjunction and implication can be derived. Two common derived path operators are: the *eventually* operator F where $\mathsf{F}^{\leq n}\, \Phi \equiv \mathit{true}\, \mathsf{U}^{\leq n}\, \Phi$ and the *always* operator G where $\mathsf{G}\, \Psi \equiv \neg(\mathsf{F}\, \neg \Psi)$. If $N = \infty$ then the superscript N is omitted. In PRISM $\mathtt{P}_{=?}[\Psi]$ computes the probability for Ψ to hold in a given state (initial state by default). $\mathtt{S}_{=?}[\Phi]$ computes the steady-state (long-run) probability of being in a state which satisfies a Boolean-valued state formula Φ. PRISM supports a *reward*-based extension of PCTL called *rPCTL* that assigns non-negative real values to states and/or transitions. $\mathtt{R}_{rwd=?}\left[\, \mathsf{C}^{\leq N} \right]$ computes the reward named *rwd* accumulated along *all* paths within N time-steps, $\mathtt{R}_{rwd=?}\left[\, \mathsf{F}\ \phi \,\right]$ computes the reward named *rwd* accumulated along *all* paths until ϕ is satisfied. Filters check for properties that hold from sets of states satisfying given propositions. Here we use **state** as the filter operator: e.g., **filter**(**state**, ϕ, *condition*) where ϕ is a state formula and *condition* a Boolean proposition uniquely identifying a state in the DTMC.

3 Probabilistic Models for Software Usage

Our experience indicates that useful usage models do *not* depend only on static attributes such as the location, gender, age of users, but on *dynamic* attributes such as *patterns* of use, that may change over time. Users are heterogeneous and each individual user may move between different patterns, from one interactive session to another, or even during an interaction session. We need new models of dynamic usage patterns and new ways to analyse those models. To encapsulate the patterns of dynamic and heterogeneous users, we define models that are:

- *probabilistic*: statistical models of the different generating processes of heterogeneous, dynamic users,
- *Markovian*: behaviour is determined by current state, not process history,
- *admixture*[1]: usage patterns (or, more generally, behavioural traits) are complex and drawn from a probability distribution, i.e. individuals move between different patterns during an observed trace.

First-order Markov models have been shown to provide a good modelling approach for human navigation on the Web [6–10] (where the states correspond to visited webpages) as well as within mobile applications [3,11] (where the states correspond to device screen events). We model the usage pattern as a hidden/latent state and the user-initiated events as observed states. Each hidden state defines a first-order discrete time Markov chain (DTMC) over observed states, which we call *activity pattern*.

Let $\mathcal{P}$ be a population of M user traces, with each user trace $\alpha^1, \ldots, \alpha^M$ a finite sequence over a set $\mathcal{A}$ of n events (labels of logged user interactions). Some

[1] Admixture models derive from genetic analysis of populations where individuals have mixed ancestry: each individual inherits a fraction of his/her genome from ancestors.

events are distinguished by the unique labels `startS` and `stopS` to denote the beginning and end (resp.) of a user *session* within a trace. Each user trace is a concatenation of sessions. It is important to note that each user trace contains a variable number of sessions, and each session is a variable length sequence of events. We now define two different usage models for a given population of user traces.

Definition 1 (Population admixture model). *For a given positive integer K, a* population admixture model *(PAM) for the user trace population $\mathcal{P}$ is a tuple $(\mathcal{X}, \mathcal{Y}, B, \mathcal{L}, \Theta)$ where:*

- *$\mathcal{X}$ is the set of latent states, $\mathcal{X} = \{1, \ldots, K\}$,*
- *$\mathcal{Y}$ is the set of observed states, $\mathcal{Y} = \{0, \ldots, n-1\}$,*
- *B is the observation probability matrix with $B : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [0, 1]$ such that for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ we have $\sum_{y' \in \mathcal{Y}} B(x, y, y') = 1$,*
- *$\mathcal{L} : \mathcal{Y} \to \mathcal{A}$ is the labelling function,*
- *$\Theta = \{\theta_1, \ldots, \theta_M\}$ is the set of distributions over all latent states for each user trace in the population, such that for all m, $1 \leq m \leq M$, $\theta_m : \mathcal{X} \to [0, 1]$ with $\sum_{x \in \mathcal{X}} \theta_m(x) = 1$.*

Definition 2 (Generalised population admixture model). *For a given positive integer K, a* generalised population admixture model *(GPAM) for the user trace population $\mathcal{P}$ is a tuple $(\mathcal{X}, \mathcal{Y}, \pi, A, B, \mathcal{L})$ where $(\mathcal{X}, \mathcal{Y}, \pi, A, B)$ is an AR-HMM and $\mathcal{L} : \mathcal{Y} \to \mathcal{A}$ is the labelling function.*

Definition 3 (Activity patterns). *For any PAM $(\mathcal{X}, \mathcal{Y}, B, \mathcal{L}, \Theta)$ or GPAM $(\mathcal{X}, \mathcal{Y}, \pi, A, B, \mathcal{L})$, and any latent state $x \in \mathcal{X}$, the tuple $(\mathcal{Y}, \mathcal{L}^{-1}(\texttt{startS}), B(x), \mathcal{L})$ is a discrete-time Markov chain we call* activity pattern.

Hidden Markov models express relationships only between latent states, but not between observed states, so they are not suitable for our purpose of analysing user-software interaction. PAM models are admixtures of first-order Markov chains (DTMCs) that express relationships between observed states, with Θ defining distributions over the latent states. PAM models are well-suited for user-intensive software, in particular because we can study the values for Θ across all users or subpopulations thereof. In the PAM models, it is not possible to relate changes to latent states to changes in the observed states. If this is important, then AR-HMMs (GPAM) are more suitable because they express explicit relationships between both observed and latent states.

Each of the models PAM and GPAM (pictorial representation in Fig. 2) offers a different perspective on usage behaviours, and consequently affords different analysis. Both are based on K mixture components (or latent states), with each component a DTMC (an activity pattern). Each component DTMC has the same set of states, but the transition probabilities are different, as indicated by the thickness of transitions. The number of mixture components/activity patterns, K, is an exploratory tool: we do not try to find the "correct" or optimal value

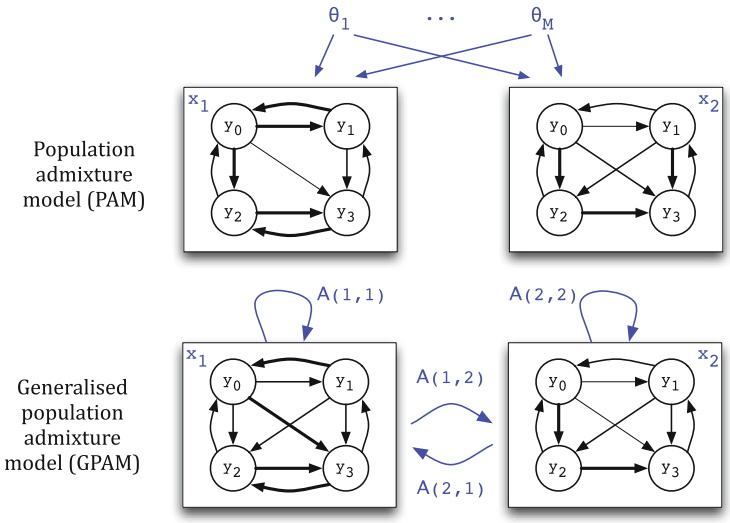


Fig. 2. Pictorial representation of PAM and GPAM for two latent states (or activity patterns as DTMCs in each box) x_1 and x_2, $K = 2$, four observed states $y_0 - y_3$, M user traces, $1 \leq m \leq M$

for K, instead we explore the variety of activity patterns that are meaningful to software evaluation (e.g. we might develop distinct versions for three or four different activity patterns).

3.1 Learning Model Parameters

The size of the latent state set $\mathcal{X}$ (the number K of the activity patterns), the set of n observed states (user-initiated events) $\mathcal{Y}$, and the labelling function $\mathcal{L}$ are pre-determined for both PAM and GPAM. It remains to define the observation matrix B and distributions Θ for PAM, and the transition matrix A, observation matrix B, and initial distribution π for GPAM. We *learn* these parameters from sets of user traces using statistical methods for maximum likelihoods.

We segment the user traces according to intervals over days, weeks, or months, based on timestamps associated with each logged event (the time stamps are then discarded). Intervals have the form $[d_1, d_2]$, which includes the user traces from the d_1^{th}-th up to the d_2^{th} day of usage.

Given a set $\mathcal{P}$ of user traces, we compute $n \times n$ transition-occurrence matrices for each trace α such that α_{ij} at position (i, j) is the number of times the subsequence α_i, α_j occurs in α. This is the input data for parameter inference.

For PAM we employ the local non-linear optimisation Expectation–Maximisation (EM) algorithm [12] for finding maximum likelihood parameters of observing each trace, restarting the algorithm whenever the log-likelihood has multiple-local maxima. EM converges provably to a local optimum of the criterion, in this case the likelihood function. For GPAM (AR-HMM) we employ

the Baum–Welch algorithm [13], which uses also the EM algorithm. We use EM, as opposed to say Markov chains Monte Carlo methods, because it is fast and computationally efficient for our kind of data.

3.2 Encoding Models in the PRISM Modelling Language

For both PAM and GPAM we analyse each activity pattern as a DTMC in PRISM. In addition, for GPAM we *flatten* the whole AR-HMM model, which has two types of states and three types of transitions (between latent states, between observed states, from latent states to observed states) into a DTMC (and subsequently analyse it in PRISM) as follows. We pair each observed state with a hidden state. The transition probability between two any such pairs of states (y, x) and (y', x') is the probability of moving from hidden state x to hidden state x' times the probability of moving between the observed states y to y' while in the current hidden state x', i.e., $\mathbf{P}((y, x), (y', x')) = A(x, x') \cdot B(x', y, y')$. The initial state $(\mathcal{L}^{-1}(\texttt{startS}), 0)$ is a dummy that encodes the global initial distribution. This flattening operation shares similarities with the composition between an HMM and a deterministic finite state machine (see [14]), however we are composing an HMM with a DTMC.

Definition 4. *Given a GPAM* $(\mathcal{X}, \mathcal{Y}, \pi, A, B, \mathcal{L})$*, we flatten it into the DTMC* $(S, (\bar{y}, 0), \mathbf{P}, \mathcal{L})$ *where:*

- $S = \mathcal{Y} \times (\{0\} \cup \mathcal{X})$*, where* $\mathcal{X} = \{1, \ldots, K\}$*,*
- $\bar{y} = \mathcal{L}^{-1}(\texttt{startS})$ *is the initial observed state with the label* `startS`*,*
- $\mathbf{P}((y, x), (y', x')) = A(x, x') \cdot B(x', y, y')$*, for* $x > 0$*,*
- $\mathbf{P}((\bar{y}, 0), (y, x)) = \begin{cases} \pi(x) & \text{if } y = \bar{y} \text{ and } x' > 0 \\ 0 & \text{otherwise} \end{cases}$
- $\mathcal{L}(y, x) = \mathcal{L}(y) \times \{x\}$.

For clarification, DTMCs are used at two levels of abstraction: to represent individual activity patterns and to represent the more complex GPAM Markov models in which the activity patterns are embedded, as a flattened GPAM.

4 Temporal Properties for Analysing Usage Models

We present *generic* rPCTL properties for analysing: (1) individual activity patterns of both PAM and (flattened)[2] GPAM models, and (2) *GPAM-specific* rPCTL and PCTL* properties that involve moving *between* activity patterns. It is important to note we cannot check properties from the latter set on PAM models because PAMs include a distribution over activity patterns and not transition probabilities between activity patterns, as in GPAMs.

[2] All results are for flattened GPAMs.

4.1 Generic Properties for Individual Patterns in PAM and GPAM

We consider the following classes of state formulae:

$$\begin{aligned} \textit{Observed state formulae} \quad & \varphi ::= \texttt{true} \mid \ell \mid y = j \mid \neg\varphi \mid \varphi \wedge \varphi \\ \textit{Hidden state formulae} \quad & \gamma ::= \texttt{true} \mid x = i \mid \neg\gamma \mid \gamma \wedge \gamma \\ \textit{Non-probabilistic state formulae} \quad & \phi ::= \varphi \mid \gamma \mid \neg\phi \mid \phi \wedge \phi \end{aligned}$$

where ℓ ranges over the set of observed state labels $\mathcal{A}$, j over the set of observed states identifiers $\{0, \ldots, n-1\}$ and i over the set of hidden state identifiers $\{0, 1, \ldots, K\}$ (with 0 denoting a dummy hidden state for the PRISM encoding). We define the following reward structures for all values of $j \in \{0, \ldots, n-1\}$ and $i \in \{1, \ldots, K\}$: `rLabelj` and `rLabeljAPi` for computing the expected number of visits to an observed state j and to a state (i, j) respectively, and `rSteps` and `rStepsAPi` for computing the number of all time steps in the model and the number of all time steps in activity pattern i respectively. Note that `rLabeljAPi` and `rStepsAPi` are defined exclusively for GPAM models. The PRISM definitions of these rewards are as follows:

```
rewards "rLabelj" (y=j): 1; endrewards
rewards "rLabeljAPi" (y=j) & (x=i) : 1; endrewards
rewards "rSteps" [] true : 1; endrewards
rewards "rStepsAPi" [] (x=i) : 1; endrewards
```

Table 1 lists three classes of generic properties: VisitProb, VisitCount, and StepCount, where the reward name *rewardV* can be either of `rLabelj` and `rLabeljAPi`, while *rewardS* can be either of `rSteps` and `rStepsAPi`. We adopt the following interpretation of the model checking results for activity patterns in the same model (either PAM or GPAM), and the same value of N, when comparing the states. We say that a pair of observed state j and activity pattern i scores a better (resp. worse) result that (j', i') if: VisitProb returns a higher (resp. lower) value, VisitCount a higher (resp. lower) value, StepCount a positive lower (resp. higher) value for the pair (j, i) than for (j', i').

Table 1. Classes of rPCTL properties applicable in PAM and GPAM model.

Prop.	Description	rPCTL formula
VisitProb	Probability that starting from state satisfying ϕ_0, ϕ_1 holds until reaching a state in which ϕ_2 holds, within N time-steps	$\mathbf{filter}(\mathbf{state}, \mathtt{P}_{=?}[\phi_1 \mathsf{U}^{\leq N} \phi_2], \phi_0)$
VisitCount	Starting from state satisfying ϕ_0, expected reward cumulated over N time-steps	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{rewardV=?}[\mathsf{C}^{\leq N}], \phi_0)$
StepCount	Starting from state satisfying ϕ_0, expected number of steps cumulated before reaching a state satisfying ϕ_1	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{rewardS=?}[\mathsf{F}\,\phi_1], \phi_0)$

Table 2 illustrates some instances of the property classes in Table 1. VP1 and VP2 are simple reachability properties for a particular state in either an activity pattern (AP) or a GPAM respectively. VP3 computes the reachability probability for a state j in pattern i while always being in pattern i in an GPAM. SC1 and SC2 compute the expected number of steps needed to reach a particular state in either an AP or a GPAM, while SC3 computes only the steps takes in the pattern i to reach a state in an GPAM. VC1 computes the expected number of visits to a state j in an AP, while VC2 computes only the number of visits to state j while the pattern i in a GPAM.

Table 2. Instances of VisitProb, VisitCount, and StepCount applicable to activity pattern (AP) i in PAM or GPAM, parametrised by observed state j and pattern i.

Prop.	DTMC	rPCTL formula
VP1	AP/PAM	$\mathbf{filter}(\mathbf{state}, \mathtt{P}_{=?}[\mathtt{true}\,\mathsf{U}^{\leq N}(y=j)], \mathtt{startS})$
VP2	GPAM	$\mathbf{filter}(\mathbf{state}, \mathtt{P}_{=?}[\mathtt{true}\,\mathsf{U}^{\leq N}(x=i \wedge y=j)], (\mathtt{startS} \wedge x=0))$
VP3	GPAM	$\mathbf{filter}(\mathbf{state}, \mathtt{P}_{=?}[(x=i)\,\mathsf{U}^{\leq N}(x=i \wedge y=j)], (\mathtt{startS} \wedge x=i))$
SC1	AP/PAM	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rSteps}=?}[\mathsf{F}\,(y=j)], \mathtt{startS})$
SC2	GPAM	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rSteps}=?}[\mathsf{F}(x=i \wedge y=j)], (\mathtt{startS} \wedge x=0))$
SC3	GPAM	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rStepsAPi}=?}[\mathsf{F}(x=i \wedge y=j)], (\mathtt{startS} \wedge x=i))$
VC1	AP/PAM	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rStatej}=?}[\mathsf{C}^{\leq N}], \mathtt{startS})$
VC2	GPAM	$\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rStatejAPi}=?}[\mathsf{C}^{\leq N}], (\mathtt{startS} \wedge x=i))$

4.2 GPAM-Specific Properties

For GPAM, it is possible to consider properties that involve multiple activity patterns. A few such rPCTL/PCTL* properties are listed in Fig. 3.

PG1 : $\mathbf{filter}(\mathbf{state}, \mathtt{P}_{=?}[(\neg\mathtt{UseStop})\,\mathsf{U}\,(x=i_1 \wedge y=j_1)], (x=i_0 \wedge y=j_0))$
PG2 : $\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rSteps}=?}[\mathsf{F}\,(y=j_1)], (x=i_0 \wedge y=j_0))$
PG3 : $\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rSteps}=?}[\mathsf{F}\,(x=i_1 \wedge y=j_1)], (x=i_0 \wedge y=j_0))$
PG4 : $\mathbf{filter}(\mathbf{state}, \mathtt{R}_{\mathtt{rStepsAPi}=?}[\mathsf{F}\,(x=i_1 \wedge y=j_1)], (x=i_0 \wedge y=j_0))$
PG5 : $\mathtt{P}_{\geq 1}[\,\mathsf{F}\,(x=i \wedge y=j)] \wedge \mathtt{P}_{\geq 1}[\,\mathsf{G}\,((x=i \wedge y=j) \Rightarrow \mathtt{P}_{>p}[(x=i)\,\mathsf{U}\,\mathtt{stopS}\,])]$
PG6 : $\mathtt{P}_{\geq 1}[\,\mathsf{F}\,\phi_1] \wedge \mathtt{P}_{\geq 1}[\,\mathsf{F}\,\phi_2] \wedge \mathtt{P}_{\geq 1}[\,\mathsf{F}\,((\neg\phi_1 \wedge \neg\phi_2)\,\mathsf{U}\,\mathtt{P}_{\geq 1}[(\phi_1 \wedge \neg\phi_2)\,\mathsf{U}\,\mathtt{P}_{\geq p}[\,\mathsf{X}\,\phi_2]])]$
PG7 : $\mathtt{S}_{=?}[\,x=i\,]$

Fig. 3. rPCTL* properties where i, i_0, i_1 range over hidden states, j, j_1, j_2 over observed states, $p \in [0,1]$.

Property PG1 is an instance of VisitProb (reasoning over reachability within the same user session), while PG2 – PG4 are instances of StepCount. PG5 helps

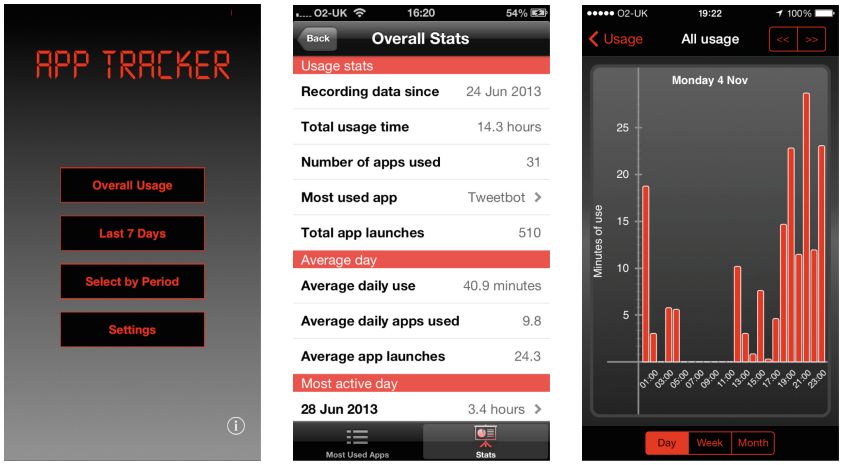
us identify most likely states and patterns that lead to the end of the session (this is useful for user engagement analysis across different time intervals). PG6 checks for correlations between two properties (that might involve states and activity patterns): if eventually one property holds (for a period of time), then immediately afterwards, the second property holds. PG7 computes the long-run probability of the population of user traces of being in each of the activity patterns and it gives us more insight into the popularity of the patterns.

5 Example Software Usage Analysis: AppTracker

AppTracker [15] is a personal productivity iOS mobile application that runs in the background, monitoring the opening and closing of (other) apps. It displays a series of charts and statistics, offering insight to users into the time spent on their device, the most used apps, how these stats fluctuate over time, etc. The main menu screen offers four main options (Fig. 4(a)). The first menu item, *Overall Usage*, contains summaries of all the data recorded since AppTracker was installed and opens the views `OverallUsage` and `Stats` (Fig. 4(b)). The second menu item, *Last 7 Days*, opens the view `StackedBars` and displays a chart indicating activity recorded over the last 7 days. The third menu item, *Select by Period*, opens the view `PeriodSelector` and shows statistics for a selected period of time. For example, this could be which apps were used most since last Saturday, or how time spent on Facebook varied each day last month, or hourly device usage for a particular day (Fig. 4(c)). The final menu option, *Settings*, allows a user to start and stop the tracker, or to reset their recorded data. We are interested in the 16 user-initiated events that switch between views, as illustrated by the state diagram in Fig. 5. These events determine the atomic propositions used in our models. The state labels denoting the start and the end of a session are `UseStart` and `UseStop` respectively. The log data is stored in a MySQL database by the SGLog framework. Raw data is extracted from the database and processed using JavaScript to obtain user traces in JSON format has the following format: information about the user's device, start and end data of AppTracker usage, and list of sessions. For example, the start of a user trace may look like the following:

```
[{"deviceid":"_","firstSeen":"2013-08-20 09:10:59","lastSeen":"2014-03-24
09:57:32","sessions":[[{"timestamp":"2013-08-20 09:11:01","data":
"UseStart"},{"timestamp":"2013-08-20 09:11:02","data":"T&C"},{"timestamp":
"2013-08-20 09:11:23", "data":"Main"},{"timestamp":"2013-08-20 09:11:46",
"data":"OverallUsage"},...]}]
```

AppTracker was first released in 2013 and downloaded over 35,000 times; our data sets are taken from a sample of 489 users traces during 2013 and 2014. For learning parameters, the EM algorithm was restarted 200 times with 100 maximum number of iterations for each restart. As example performance, we implemented the algorithm for GPAM parameters in Java and ran it on a 2.8 GHz Intel Xeon (single thread, one core); for the first week of logged usage data, the



(a) Main menu (b) Overall stats (c) Daily device usage

Fig. 4. Screenshots from AppTracker.

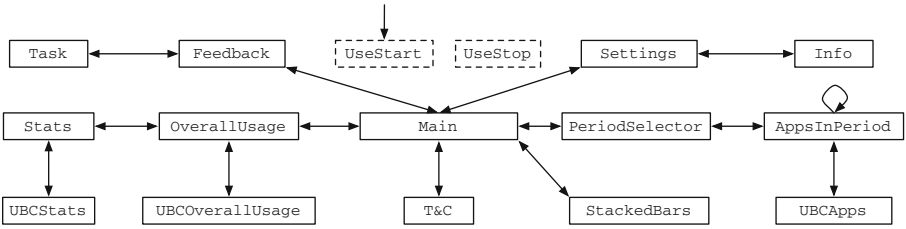


Fig. 5. AppTracker state diagram. `UseStart` labels the initial state; all (15) outgoing and incoming arrows are omitted for `UseStart` and `UseStop`.

algorithm takes 1.7 min for $K = 2$, 2.5 min for $K = 3$, and 3.7 min for $K = 4$. Note that complexity of the inference depends on the n and value of K, not on the size of the data sets.

Analysis of individual patterns. This helps us identify distinct characteristics of each pattern and give patterns names that are meaningful. In [2] we considered five properties for PAM: the ones listed in Table 2 and two instances of the VisitProb and StepCount (where the starting state corresponds to a screen view different from `UseStart`). Here, as an example of properties for GPAM, we give results for VP2 and VC2 in Table 3 for $K = 3$. For the first month of usage, AP1 has the best results for `Stats` and almost the worst for the other states, AP2 has the best results for `OverallUsage`, poor results for `StackedBars` and `PeriodSelector`, and worst for `Stats`, while AP3 has the best results for `StackedBars` and `PeriodSelector`, however `OverallUsage` scores better, and `Stats` has also good results. Therefore for the first month, we conclude that AP1 identifies activity around `Stats` only, AP2 is a Glancing pattern (based around the screen view `OverallUsage`), and AP3 is an Overall Viewing pattern (based

around high level exploration of the app). For the second month of usage we see different results. AP1 is more likely a Glancing pattern, AP2 an Overall Viewing pattern, and AP3 an In-depth Viewing pattern (based around in-depth visualisations of usage statistics for specific periods of interest). For the third month, AP1 is an In-depth Viewing pattern, AP2 is an Overall Viewing pattern, and AP3 is a Glancing pattern.

We draw two main conclusions: (1) during the first month there is an exceptional activity around `Stats` screen view compared to the later months (we will see additional information revealed by the analysis of PG5); (2) during the second month and the third month of usage we can identify a distinctive Glancing pattern characterised by frequent visits to `OverallUsage` and, surprisingly, to `StackedBars`, though that is almost 5 times less frequent. These interpretations, based of the results listed in Table 3, gives only a glimpse of the characteristics of the patterns. More information is obtained by analysing the other properties and correlating the results.

Table 3. GPAM results for properties VP2 (probability to reach screen view in a particular activity pattern within 50 time steps) and VC2 (expected number of visits to a screen view only in a particular activity pattern within 50 time steps) for $K = 3$ and j taking state identifiers values associated to the labels `OverallUsage`, `StackedBars`, `PeriodSelector`, and `Stats`. Colour-coded interpretation for a property, screen view, and time cut across all 3 patterns: red results are the best, followed by the blue ones, and black is for the worst result. The meaning of each AP needs to be determined by the evaluator for each time cut as they may be different.

Time cut	Prop.	`OverallUsage`			`StackedBars`			`PeriodSelector`			`Stats`		
		AP1	AP2	AP3	AP1	AP2	AP3	AP1	AP2	AP3	AP1	AP2	AP3
1st month	VP2	0.24	0.99	0.85	0.19	0.30	0.75	0.28	0.24	0.54	0.75	0.01	0.68
	VC2	0.52	6.23	1.98	0.21	0.36	1.44	0.37	0.49	0.79	1.66	0.01	1.17
2nd month	VP2	0.73	0.98	0.23	0.00	0.76	0.66	0.04	0.14	0.64	0.21	0.23	0.79
	VC2	2.47	5.52	0.26	0.00	1.55	1.19	0.05	0.15	1.79	0.25	0.33	1.99
3rd month	VP2	0.39	0.40	0.97	0.60	0.21	0.69	0.46	0.48	0.29	0.67	0.20	0.33
	VC2	0.55	1.38	5.63	1.06	0.31	1.26	0.72	1.07	0.67	1.36	0.55	0.48

Analysis specific to GPAM. We analysed 18 GPAM models ($K \in \{2, 3, 4\}$; 6 time cuts) using the properties of Table 2 and PG1–PG7 instantiated for various combinations of states and activity pattern identifiers. The results confirmed that AppTracker has three major activity patterns: Overall Viewing, In-depth Viewing, Glancing, moreover the patterns do not correlate with the top-level menu structure (we observe this when we consider larger values for K). In addition, the GPAM-specific properties PG1–PG4 bring new information compared to the PAM analysis such as: (a) finer insight into each pattern characteristics in relation to the other patterns and (b) a stronger case for a distinct activity pattern, Glancing, for short interactions around the `OverallUsage` screen view, especially for time intervals excluding the first few days of usage.

Due to space constraints, we give details for only one property, PG5, which concerns user disengagement. This property computes the probability that *always from a screen view j in a pattern i, eventually, without changing the pattern, the session ends.* Analysing this property allows us to identify most likely screen views and patterns that diminish user engagement (i.e. lead to end of the session). Table 4 lists the results for PG5, two cases, $K = 2$ and $K = 3$. For the first month of usage (and this also holds for all time intervals starting from first day of usage), the views `Stats`, `UBCOverall` and `UBCApps` are the most likely to disengage users in Glancing and Overall Viewing patterns where the probabilities p to do so are the highest over all states (see in bold), while in the In-depth Viewing pattern PG5 either does not hold, or it holds for close to zero values of p. While for the views `UBCOverall` and `UBCApps` this effect was expected because of their lowest level in the hierarchical menu, we were intrigued by the results for `Stats`. We discussed these results with the designers of AppTracker and found out that `Stats` does not show much information up until the first 30 days of usage; therefore they subsequently suggested changing the monthly `Stats` view to show something engaging (e.g. default illustrations of monthly stats and/or quick tutorial) up until the first 30 days of usage so that users do not get put off and end the session. During the second month of usage, for $K = 2$ the probabilities p are similar for both patterns. For $K = 3$, during the first month of usage `Stats` is most likely across all screen views to disengage users, whereas during the second month we see other screen views such as `OverallUsage` and `StackedBars` as likely as `Stats` to disengage users.

Table 4. GPAM results for property PG5, for $K = 2$ and $K = 3$, and first and second months of usage. The two values in each cell are the probability bound p for the first month and the second month of usage respectively such that PG5 holds; if no value for p is given, the property does not hold for any value of p.

State label	K = 2		K = 3		
	Overall	In-depth	Overall	In-depth	Glancing
`OverallUsage`	0.39/0.63	–/0.69	0.55/0.17	–/0.08	0.33/0.87
`StackedBars`	0.46/0.63	–/0.76	0.55/0.66	–/0.01	0.37/–
`PeriodSelector`	0.26/0.43	–/0.62	0.37/0.05	–/0.09	0.14/0.51
`AppsInPeriod`	0.15/0.14	0.07/0.56	0.28/0.03	0.06/0.01	0.12/0.32
`Stats`	**0.71**/0.59	–/0.68	**0.68**/0.22	0.01/0.13	**0.62**/0.86
`UBCOverall`	**0.81**/0.56	–/–	0.38/–	0.02/0.02	0.08/0.77
`UBCApps`	**0.84**/0.14	0.01/–	**0.61**/0.02	–/0.05	0.47/–

6 Discussion

Which models, which temporal properties. PAM and GPAM analyses with generic properties and models generated from different usage intervals pro-

vide insight into the main characteristics of the patterns. We suggest this is a good starting point for any study of usage patterns. This is followed by GPAM-specific analysis, which allows us to reason about changing patterns within a temporal property. This analysis may be focused around interesting characteristics revealed during the initial, generic analysis (e.g. the unusual activity around `Stats` view). A feature of PAM is the user trace distribution: this informs us about which user traces are "interesting" to investigate further. Here we gave only PAM and GPAM results, ideally we would also pick certain individual user traces (e.g. those engaged for a long/short time period), define sub-populations (according to ranges of θ), and analyse the relevant models.

Temporal aspects. Our approach has two temporal aspects: data logging intervals and path formulae in the generated model, they should not be confused as they are at different scales.

Effect of parameter inference. Models depend on the data and inferred parameters, so property checking will give different results for different models (and different runs of the inference algorithm), but we expect to see similar ratios between sets of learnt parameters.

Longitudinal and other types of analysis. Our example indicates results may be sensitive to the chosen time interval for logged data. Common sense indicates this is to be expected, for example differences between the first day of use and use after several weeks. However, there may be software-specific temporal sensitivities: our example indicated an unanticipated sensitivity in the first month of use. We note there are other categorisations: device type, timezone, length of user trace in number of sessions, frequency of sessions, and average length of sessions, to name a few. The plethora of possible categorisations and parametrised properties can be overwhelming and even more so the consideration of how to present the analysis results (e.g. as tables, as plots). Systematic approaches to categorisation and interpretation of results are needed.

Related work. User behaviours are considered in [10] where models are based on static user attributes rather than on inferred behaviours, assuming within-class use to be homogeneous, whereas we demonstrate within-class variation. Zang et al. [16] extended probabilistic model checking to HMMs, this is not applicable as we are interested in properties involving both latent and observed states; moreover, we do not analyse properties directly on AR-HMMs, but on their flattened version as DTMCs. The two hidden Markov models we use, PAM and GPAM, are examples of dynamic Bayesian networks (DBNs) [5,17]. DBNs can be analysed using Bayesian statistical model checking [18], however our models are easily encoded and analysed using probabilistic model checking. Extensive works on inferring parameters for first-order HMM models (as DBNs) from user traces for runtime verification purposes can be found in [14,19], however we are looking at a different class of HMM, namely admixture models, for identifying and analysing usage models within a population of user traces.

7 Conclusions and Future Work

We have defined new models of software usage based on two new admixture, latent variable Markov models: PAM and GPAM. Both models are admixtures of activity patterns (DTMCs) and are generated from logged data sets of interactions of actual deployments, over different time intervals. Temporal logic property analysis is by model checking. We defined two sets of (parametrised) temporal logic properties: generic ones for analysis of individual activity patterns and GPAM-specific for properties that combine patterns. The generic properties provide insight into the main characteristics of the patterns, and we suggest this is a good starting point for any study of usage patterns. This is followed by GPAM-specific analysis, which may be focused around interesting or unusual results from the generic analysis. Application to logged data from a real-life interactive software system indicates that out approach is tractable and useful; future work includes guidance on how and when to investigate each of the models, and a systematic approach to interpretation of results.

Acknowledgement. This research is supported by EPSRC Programme Grant *A Population Approach to Ubicomp System Design* (EP/J007617/1).

References

1. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
2. Andrei, O., Calder, M., Chalmers, M., Morrison, A., Rost, M.: Probabilistic formal analysis of app usage to inform redesign. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 115–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33693-0_8
3. Andrei, O., Calder, M., Higgs, M., Girolami, M.: Probabilistic model checking of DTMC models of user activity patterns. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 138–153. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10696-0_11
4. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
5. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press, Cambridge (2012)
6. Borges, J., Levene, M.: Data mining of user navigation patterns. In: Masand, B., Spiliopoulou, M. (eds.) WebKDD 1999. LNCS (LNAI), vol. 1836, pp. 92–112. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44934-5_6
7. Chierichetti, F., Kumar, R., Raghavan, P., Sarlós, T.: Are web users really Markovian? In: Mille, A., Gandon, F.L., Misselis, J., Rabinovich, M., Staab, S. (eds.) Proceedings of the 21st World Wide Web Conference 2012 (WWW 2012), pp. 609–618. ACM (2012)
8. Singer, P., Helic, D., Taraghi, B., Strohmaier, M.: Detecting memory and structure in human navigation patterns using Markov chain models of varying order. PLoS One **9**(7), 1–21 (2014)

9. Singer, P., Helic, D., Hotho, A., Strohmaier, M.: HypTrails: a Bayesian approach for comparing hypotheses about human trails on the web. In: Gangemi, A., Leonardi, S., Panconesi, A. (eds.) Proceedings of the 24th International Conference on World Wide Web (WWW 2015), pp. 1003–1013. ACM (2015)
10. Ghezzi, C., Pezzè, M., Sama, M., Tamburrelli, G.: Mining behavior models from user-intensive web applications. In: Proceedings of the 36th International Conference on Software Engineering (ICSE 2014), pp. 277–287. ACM (2014)
11. Kostakos, V., Ferreira, D., Gonçalves, J., Hosio, S.: Modelling smartphone usage: a Markov state transition model. In: Lukowicz, P., Krüger, A., Bulling, A., Lim, Y., Patel, S.N. (eds.) Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016), pp. 486–497 (2016)
12. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc.: Ser. B (Methodol.) **39**(1), 1–38 (1977)
13. Welch, L.: Hidden Markov models and the Baum-Welch algorithm. IEEE Inf. Theory Soc. Newslett. **53**(4), 10–13 (2003)
14. Bartocci, E., Grosu, R., Karmarkar, A., Smolka, S.A., Stoller, S.D., Zadok, E., Seyster, J.: Adaptive runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 168–182. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_18
15. Bell, M., Chalmers, M., Fontaine, L., Higgs, M., Morrison, A., Rooksby, J., Rost, M., Sherwood, S.: Experiences in logging everyday app use. In: ACM Proceedings of Digital Economy 2013 (2013)
16. Zhang, L., Hermanns, H., Jansen, D.N.: Logic and model checking for hidden Markov models. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 98–112. Springer, Heidelberg (2005). https://doi.org/10.1007/11562436_9
17. Sucar, L.E.: Probabilistic Graphical Models: Principles and Applications. ACVPR. Springer, London (2015). https://doi.org/10.1007/978-1-4471-6699-3
18. Langmead, C.J.: Generalized queries and Bayesian statistical model checking in dynamic Bayesian networks: application to personalized medicine. In: Proceedings of CSB 2009 (2009)
19. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15