Case Study: Claroline

Claroline is an open-source web-based application dedicated to learning and online collaborative work. The instance of Claroline at University of Namur (Webcampus) is the main communication channel between students and lecturers. Students may register to courses and download documents, receive announcements, submit their assignments, perform online exercises, etc.

We reverse enginneer the different behavioural models from the anonymized Apache access log provided by the IT support team of the university. This file (5,26 Go) contains all the HTTP requests made to access the Claroline instance from January 1st to October 1st 2013. We chose to consider only requests to access PHP pages accessible to users. It represents a total of 12.689.030 HTTP requests. We setup an automated way to infer the usage model from the log, as it would be intractable to do so manually. Each entry of the file represents a single access to a page: we had to reconstruct the behaviour of a user by grouping requests.

Feature Diagram

The FD was built manually by inspecting a locally installed Claroline instance (Claroline 1.11.7). The FD describes Claroline with three main features decomposed in sub-features: User, Course and Subscription. Subscription may be open to everyone (opt Open Subscription) and may have a password recovery mechanism (opt Lost Password). User corresponds to the different possible user types provided by default with a basic Claroline installation: unregistered users (UnregisteredUser) who may access courses open to everyone and registered users (RegisteredUser) who may access different functionalities of the courses according to their privilege level (Student, Teacher or Admin). The last main feature, Course, corresponds to the page dedicated to a course where students and teacher may interact. A course has a status (Available, AvailableFromTo or Unavailable), may be publicly visible (PublicVisibility) or not (MembersVisibility), may authorize registration to identified users (AllowedRegistration) or not (Registration Denied) and may be accessed by everyone (FreeAccess), identified users (IdentifiedAccess) or members of the course only (MembersAccess). Moreover, a course may have a list of tools (Tools) used for different teaching purposes, e.g., an agenda (opt CourseAgenda), an announcement panel (opt CourseAnnoucements), a document download section where teacher may post documents and students may download them (opt CourseDocument), an online exercise section (opt CourseExercise), etc. Since we are in a testing context, one instance of the FD does not represent a complete Claroline instance, but the minimal instance needed to play a test-set (i.e., test cases formed as sequences of HTTP requests). Basically, it will correspond to a Claroline instance with its subscription features, one particular user and one particular course. In order to represent a complete Claroline instance, we need to introduce cardinalities on the Course and User features. Eventually we obtained a FD with 44 features.

The feature diagram in TVL format (see TVL) may be downloaded here: claroline.tvl

The feature diagram in SPLOT format (see SPLOT may be downloaded here: claroline.splot.xml

Featured Transition System

We reverse engineer the FTS using different techniques :

- In the first version, we we employed a web crawler (i.e., a Python bot that systematically browse and record information about a website) on a local Claroline instance to discover all reachable pages. Transitions have been added in such a way that every state may be accessed from anywhere. This simplification, used only to ease the FTS building, is consistent with the Web nature of the application. Moreover, some user traces show that the navigation in Claroline is not always as obvious as it seems (e.g., if the users access the website from an external URL sent by e-mail). Finally, transitions have been tagged manually with feature expressions based on the knowledge of the system (via the documentation and the local Claroline instance). To simulate a web browser access, we added a "0" initial state connected to and accessible from all states in the FTS. The final FTS consists of 107 states and 11236 transitions. The model may be downloaded here: claroline-pow.fts

Usage Model

We used different techniques to build different usage models:

- First, we used a 2-gram one without smoothing (we do not consider unseen 2-grams as we focus only on observed behaviour). Concretely, each page of the Apache Web Log is mapped to a state and each trace is completed so that it ends in the start state. The model may be downloaded here: claroline.usagemodel
- We also used the 2-gram method proposed by Sprenkle et al. (http://dx.doi.org/10.1002/stvr.1496) with Request Result (RR) and Request Result parameter Name (RRN). The models may be downloaded here: claroline-RR.usagemodel and claroline-RRN.usagemodel

Download

- **Towards Statistical Prioritization for Software Product Lines Testing**. *Devroey, X., Perrouin, G., Cordy, M., Schobbens, P., Legay, A., & Heymans, P. (2014).* In A. Wasowski & T. Weyer (Eds.), Proceedings of the 8th International Workshop on Variability Modelling of Software-intensive Systems. Nice, France: ACM. doi:10.1145/2556624.2556635

  The following archive contains all the files and tools needed to execute the trace prioritization. Please read the contained *README.txt* file to get information on how to run the prioritization : Download files and tools