A model for Pantoto core

Table of Contents

1 Introduction

1.1 Motivation and objective

We propose a model for pantoto core. The motivation for this exercise is to understand abstractly the essence of Pantoto's design. The model described here, however, is self contained and independently developed. While selectively borrowing elements of Pantoto's architecture, it is unencumbered by the compulsions of Pantoto's current design.

1.2 Core Elements

Core pantoto consists of three primary entities together referred to as the **system**: users, fields and pagelets. Users have state. Fields have content. Pagelets have views. The view depends on the user viewing the pagelet and that user's state. A user's view of a pagelet consists of a subset of fields and their permissions, ie., whether they are readable and/or writable.

Here is a simple example to ground some of the terminology and further motivate the model. The example is developed in detail later in the document. An examination process consists of two users: the student and the teacher. At any given time, the student is in one of the following possible states: ready to take the exam, writing

the exam, finished taking the exam. The examination paper is a pagelet that starts off with a view consisting of one field, the question, visible to both the teacher and the student. When the student receives the examination paper, the view changes to include the student's answer field. This field is writable by the student, readable-only by the teacher. When the student is done with the examination paper, the view changes to make the answer field read-only for the student as well.

1.3 Work flow driven evolution of pagelet views

The **workflow** of the system specifies how the states of the users evolve based on **events**. Events are atomic and instantaneous and cause the users' state to change. E.g., a student in ready state, upon receiving the exam question paper from the teacher (a 'paper' event), begins writing the answer, i.e, transitions to a 'writing' state. Events could be synchronised (as in a hand shake) or autonomous, involving only a single user. (These latter type of events are often referred to as hidden transitions.) We use Milner's CCS equations to describe workflows as processes.

The **process state** of the system at any given instant consists of the set of users and their states with respect to the system workflow. For example, the workflow in the previous example proceeds through the following process states:

```
(s_ready, t_ready)     --receive-paper-->
(s_writing, t_waiting) --submit-->
(s_waiting, t_grading) --grade-->
(s_done, t_done)
```

The set of fields and their content define the **content state** state of the system. The **view state** of the system at any instance is the set of pagelets and their views. Workflow events potentially effect changes in the view state of the system.

Fields exist independent of pagelets, although

they need to be attached to a pagelet to be viewed. A field may be attached to multiple pagelets. E.g., the question field may be part of multiple exam papers delivered to multiple students.

The **trace** of a system is sequence consists of a sequence of triples indexed by event occurrences. Each triple consists of of the system's process state, its content state and the view state immediately after the occurrence of the event.

The specification of a Pantoto core application consists of workflow specification along with rules on how the system's views change when events occur. At this time, we do not have a formal way of specifying the rules. Instead, we will illustrate several examples of system traces.

2 Type structure

Core pantoto has the following TYPES:

```
User    : TYPE
State   : TYPE
Event   : TYPE
Content : TYPE     ;; type of the content of fields
Field   : TYPE
Pagelet : TYPE
Perm    : TYPE = {--, r-, -w, rw}
```

Content is the type of possible values in the content of fields. In the examples, we use primitive types like booleans, strings, and numbers as part of Content.

3 Dynamics

The system is defined by the following three primary state variables:

```
users   : set[User]     ;; the set of users in the system
fields  : set[Field]    ;; the set of fields in the system
pagelets: set[Pagelet]  ;; the set of pagelets in the system
```

and the following secondary state variables

```
user-state    : users -> State
```

```
field-content : fields -> Content
pagelet-view  : pagelets -> users -> [fields -:> Perm]
```

In pagelet-view, the -:> indicates a partial function. Each user sees a subset of the fields and their permissions.

4 Examples:

4.1 Exam paper and grading

There are two users s and t (student and teacher) and one exam paper pagelet p, which starts off with one field q, the question. When the student receives the paper, it has an answer field a added to it. Finally, when the student submits the paper to the teacher, p acquires a grade field, which is filled and returned to the student. At each stage the workflow specifies the users, fields, pagelets and their views.

```
s_ready     = ?paper . s_writing    ;; receives paper, starts writing.
s_writing   = !submit . s_waiting   ;; submits paper, waiting for grade.
s_waiting   = ?grade . s_done       ;; receives grade, done.

t_setting   = !paper . t_waiting    ;; hands over paper, waiting.
t_waiting   = ?submit . t_grading   ;; receives paper, grading
t_grading   = !grade . t_done       ;; hands over grade, done.


s_ready | t_ready
```

The system starts off with a ready (but nervous) student and a teacher ready with the question.

We now trace the evolution of the system's state variables, which change state at each event. The following state variables are irrelevant to the discussion below and have therefore been omitted (although a complete description must include their values).

4.1.1 Initial state of the system:

```
users = {s, t}   ;; unchanged over the entire workflow.
fields = {q}
pagelets = {p}   ;; unchanged over the entire workflow.

pagelet-view(p)   = {t : {q : rw}}
```

The teacher is still setting the question paper. The student can't see the question paper at all. The pagelet's view does not have a mapping for user **student**.

4.1.2 After student receives the examination paper (paper event):

```
fields = {q, a}
pagelet-view(p)   =
 {s : {q : r-, a : rw},
  t : {q : r-, a : --}}
```

The answer is visible (and writable) only to the student. The teacher knows that there is a field called answer, but can't see it. (A different pagelet-view could allow him to view the answer as the student is writing it. Yet another pagelet-view could render the field answer completely invisible to the teacher.)

4.1.3 After the student submits the paper (submit event):

```
fields = {q, a, g}
pagelet-view(p)   =
 {s : {q : r-, a : r-},
  t : {q : r-, a : r-, g: rw}}
```

The answer field of the student is no longer writable by the student. The answer field is now visible but not writable by the teacher. A grade field is added, but only the teacher can see and manipulate it.

4.1.4 After the paper is graded (grade event):

```
pagelet-view(p)   =
  {s : {q : r-, a : r-, g: r-},
   t : {q : r-, a : r-, g: r-}}
```

The grade once given is no longer editable, even by the teacher.

The pagelet evolution could have been defined in the more "traditional" way, where all three fields of the pagelet are defined and visible from the beginning, but are selectively writable by the student or the teacher. We illustrate this below.

Note that the workflow (control) specification is the same as before.

Note that in the above simulation, we only tracked the changes in the view of the system: its users, fields, pagelets and the pagelet views. We did not need keep track of the contents of the fields.

4.2 Info pages: public views and private annotations

This example simulates an office intranet, with two users A and B. The set of pagelets consists of one pagelet, which is A's info page. It has 3 fields: his name, email, and mobile. A has rw access to each of these fields. Other intranet users have read access to these fields. An external user C only knows about the name and email; she has no knowledge of the field called mobile.

At the next event, B privately annotates A's info page. His annotation is not visible to anyone else.

The workflow dynamics of the info pages system is described by the following equation.

```
b = annotate . b

a | b | c
```

4.2.1 Initial state

```
users = {a,              ;; intranet user
         b,              ;; intranet user
         c}              ;; external user

fields = {an, ae, am}    ;; a's name, email, mobile

pagelets = {ai}          ;; a's info pagelet

pagelet-view(ai) =
  {a :  {an:rw, ae:rw, am:rw},
   b :  {an:r-, ae:r-, am:r-},
   c :  {an:r-, ae:r-}}
```

4.2.2 After B adds annotation

The users and pagelets sets remain unchanged. The updates on fields and pagelet-view is listed below:

```
fields = {an, ae, am, ba}              ;; B's private annotation added
pagelet-view(ai) =
  {a :  {an:rw, ae:rw, am:rw},
   b :  {an:r-, ae:r-, am:r-, ba:rw}, ;; B's private annotation
   c :  {an:r-, ae:r-}}               ;; is visible only to B
```

5 Discussion

It is tempting to draw an analogy between file permissions in a file system and field permissions in Pantoto core. Unlike in a file system, however, a user in Pantoto core is not endowed with read-write permissions on fields per se. Nor does the user 'own' a field or a pagelet. Similarly, the concept of a user having a permission to add or delete a field from a pagelet is not supported in this model. Rather, a user's view of a pagelet, and thereby the read-write privileges on fields, is derived from that user's role and his state in a particular workflow. In a real application, any given user is associated with multiple workflows. Since fields are shared, a user may, at a given time, have different permissions on the same field, each set of permissions tied to a different workflow. In the model described here, we do not invoke the concept of a role; that will be addressed in an extended model later.

6 Bells and Whistles

The system could support a predefined pagelet called PAGELETS which shows, for each user, pagelets that that user is allowed to see and modify. For the sake of simplicity, this special pagelet has not been included in the examples.

7 Summary

The key points are summarised below:

- A system consists of users, fields and pagelets
- Users have state, governed by a system workflow
- Workflows drive the dynamics of pagelet views

8 Implementation ideas

8.1 DSL for workflow

The DSL specifies a workflow. A syntax like that in the above examples could be a starting point.

The DSL could either be interpreted or compiled into a set of functions, each of which corresponds to a transition.

A set of users. For each user there is a list of workflows that the user is currently part of. For each workflow, there is a state that the user in relative to that workflow.

9 Todo

9.1 Examples

Include more examples, ideally taken from some existing Pantoto based sites, or e-governance applications.

9.2 Specification of view dynamics

Define a mini domain specific language to specify the pagelet view dynamics with respect to work flow events.

9.3 Extending the model

Include concept of category (this now seems like work flow), groups (roles?), and templates for agglomeration of fields.

9.4 Comparison with Pantoto implementation

Compare with current implementation. This will help improve the model, and perhaps also the implementation.