Complex Validations

A Solution using the Double-Dispatch Pattern

Background

- User-Profile model objects need validation
- Initially used standard Rails approach:

```
validates_presence_of      :password
validates_presence_of      :password_confirmation
validates_length_of        :password, :within => 4..40
validates_confirmation_of :password
validates_presence_of      :login
validates_length_of        :login, :within => 3..40
validates_uniqueness_of    :login, :case_sensitive => false
```


Selection_options_for

```
selection_options_for :status_option,
    [:pending,      'P', 'Pending'],
    [:active,       'A', 'Active'],
    [:inactive,     'I', 'Inactive'],
    [:prelicense,   'L', 'Never Logged In']
```

if user.status_option_active? ...
user.status_option_active!

Certification combinations

```
selection_options_for :cert_option,
    [:none,              'N', 'None'],
    [:csm,               'M', 'CSM'],
    [:csp,               'P', 'CSM, CSP'],
    [:cst,               'T', 'CSM, CSP, CST'],
    [:icst,              'I', 'CSM, CSP, ICST'],
    [:csp_csc,           'C', 'CSM, CSP, CSC'],
    [:csc,               'U', 'CSC'],
    [:csm_csc,           'V', 'CSM, CSC'],
    [:csm_csc_icst,      'X', 'CSM, CSC, ICST'],
    [:csp_csc_cst,       'Y', 'CSM, CSP, CSC & CST'],
    [:csp_csc_icst,      'Z', 'CSM, CSP, CSC & ICST']
```


Validate based on Status and Certification

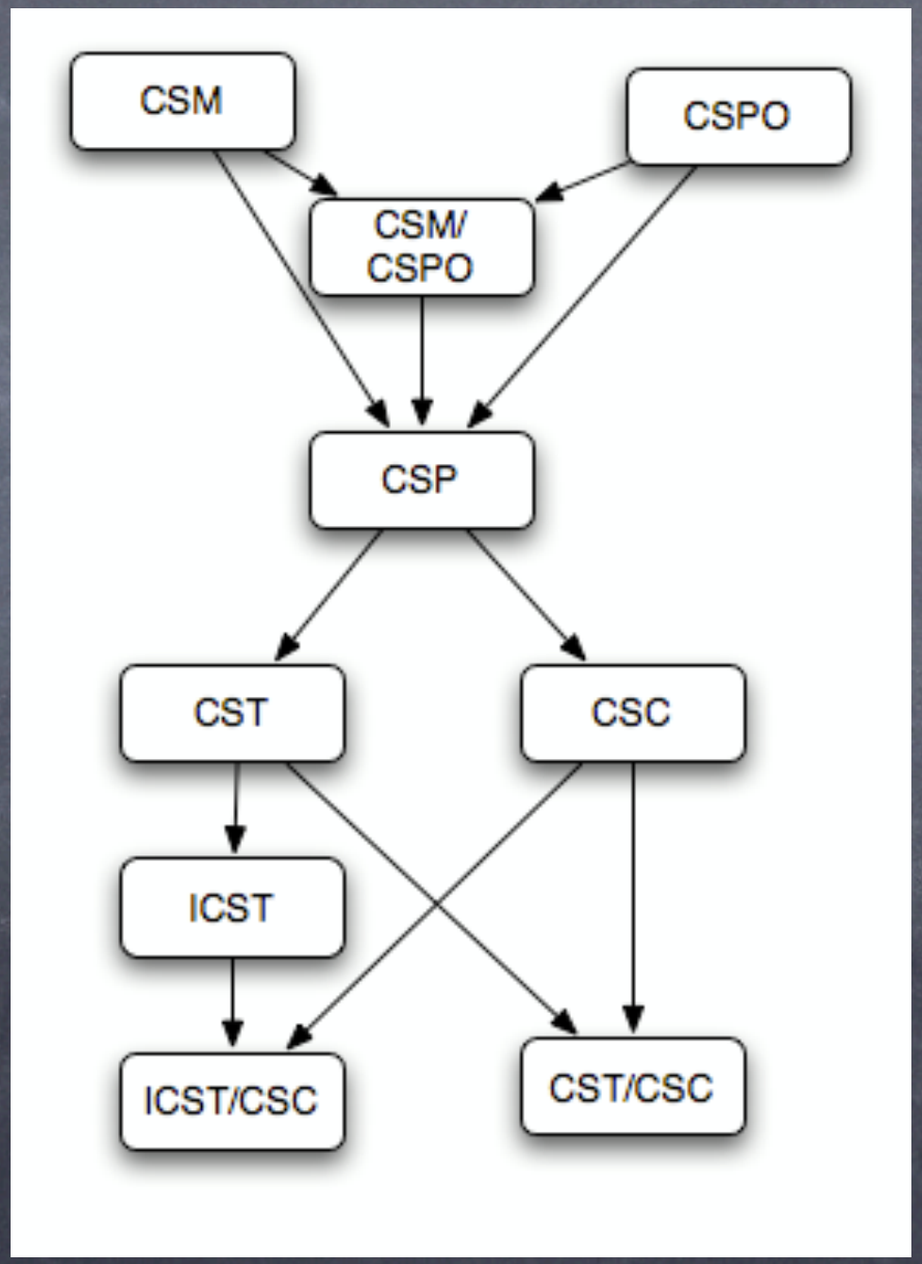
```
# class User
def validate
   case self.status_option
     when 'P': validate_status_pending
   end
end

def validate_status_pending
  if profile.accepted_csm_license_at
    errors.add(:accepted_csm_license_at, "may not be set while: 'Pending'")
  end
  if profile.accepted_cspo_license_at
    errors.add(:accepted_cspo_license_at, "may not be set while: 'Pending'")
  end
end
```


Help for data Cleanup

```
def missing_fields
    result = []
    result << 'CSM_cert_on'   if self.role.csm? && profile.csm_cert_on.nil?
    result << 'csp_cert_on'    if self.role.csp? && profile.csp_cert_on.nil?
    result << 'CST_cert_on'   if self.role.icst? && profile.cst_cert_on.nil?
    result << 'CSPO_cert_on' if self.role.po?   && profile.cspo_cert_on.nil?
    result << 'csc_cert_on'    if self.role.csc?  && profile.cspo_cert_on.nil?
    # ...
    result.uniq.join(", ")
  end
```


Certification Roles



A Growing Mess

- 33 Roles based on Certification combinations
- Roles kept as symbols (:csm, :csm_csp, etc ...)
- 4 Status (Pending, Active, Inactive, Pre-license)
- Need for Warning and then Strict Validation
- Lot of error prone conditional Logic

Create Role and Status Objects

- 33 Role objects
- 6 Status Object
  - Status, Active, Inactive, Pending, Prelicense, NullStatus

User.validate

```
def validate
  lookup_role
  status_instance . validate_errors(errors)
end

# Selection Options For ==================
def status_instance
  case status_option
    when 'P' : Status::Pending.new(self)
    when 'A' : Status::Active.new(self)
    when 'I' : Status::Inactive.new(self)
    when 'L' : Status::Prelicense.new(self)
    else Status::NullStatus.new(self)
  end
end
```


Status::Status

```
def validate_errors(visitor)
    # Wrap visitor
    def visitor.warn(fld, ignore);    end
    def visitor.error(fld, message); self.add(fld, message); end
  validate(visitor)
end
```

Status::Active

```
  def validate(visitor)
    user.role.validate_active(user, visitor)
    visitor
  end
```

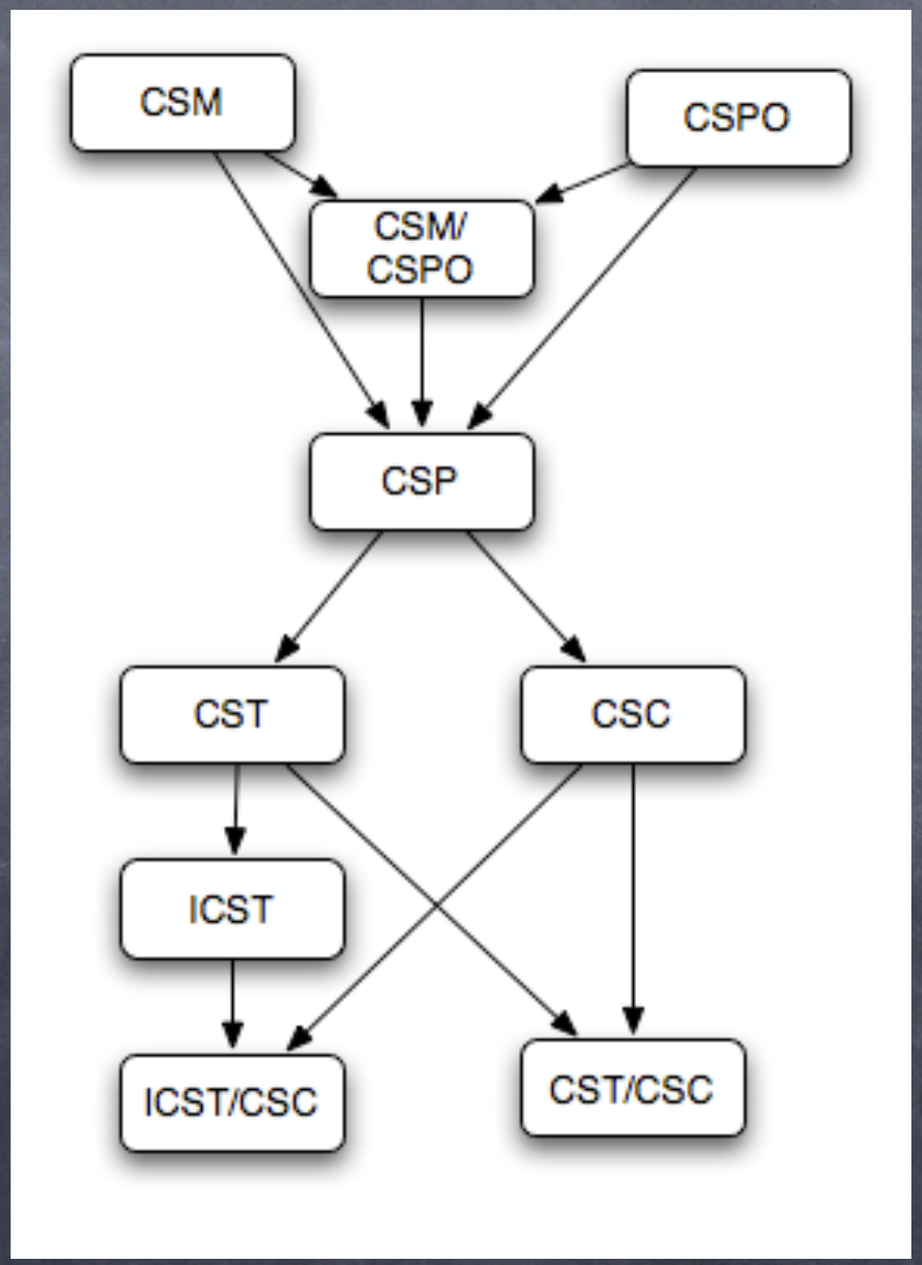
Status::Pending

```
def validate(visitor)
  user.role.validate_pending(user, visitor)
  visitor
end
```

Role::Csm

```
class Role::Csm <  Role
  def validate_active(user, visitor)
    super(user, visitor)
    p = user.profile
    return visitor unless p
    visitor.error( :csm_cert_on, 'csm_cert_on')  if p.csm_cert_on.nil?
    visitor.error( :csm_course,  'csm_course')  if p.csm_course_id.nil?
    if !p.accepted_csm_license_at &&
        p.csm_cert_on &&
        p.csm_cert_on < USER_DATA_BULK_LOADED_ON
      visitor.warn( :accepted_csm_license_at, 'accepted_csm_license_at')
    end
    visitor
  end
```


What about multiple-inheritance?



Use Modules & Meta

```
class Role::CsmCsp <  Role::Csm
      include Role::CspModule
end
```

- This adds methods and adds the new methods to a list.

Role::CspModule

```
module Role::CspModule

  def validate_active_csp(user, visitor)
    p = user.profile
    if p.csp_cert_on.nil?
      visitor.warn( :csp_cert_on, 'csp_cert_on')
    end
    if !p.csp_cert_on.nil? && (p.csp_cert_on == p.csm_cert_on)
      visitor.warn( :csp_cert_on, 'same_date_for_csP_&_csM_cert')
    end
    visitor
  end

  def self.append_features(base)
     super
     base.read_inheritable_attribute(:validate_methods) << :validate_active_csp
  end
end
```


Role::Role

```
def validate_active(user, visitor)
  # call the methods included via Role::xxxModules
  self.class.read_inheritable_attribute(:validate_methods).each do |method_sym|
    self.send( method_sym, user, visitor )
  end
  visitor
end
```

