Delegation of Obligations

Andreas Schaad & Jonathan Moffett
Department of Computer Science
University of York, UK
Outline of Talk

- Introduction and Motivation
- The Alloy Specification Language
- Organisational Obligations
- General Delegation Properties
- Delegation of Policy Objects
- Obligations require Authority
- Organisational Control through Review
- Discussion of Related Work
Introduction & Motivation (1)

- Policies express the control requirements of an organisation.
- Obligations are one specific type of policy, regulating which activities have to be performed by whom and when.
- In order to discharge his obligations a principal also needs sufficient authority expressed in authorisation policies.
Introduction & Motivation (2)

PolicyObject

Authorisation

Obligation
Introduction & Motivation (3)

- Organisational activities can be categorised according to criteria such as similarity, regularity, repeatability, etc.
- The higher this degree the more can these be regulated by policies
- The lower this degree the more individual decisions have to be made
The aim is to establish an organisation:
- Where policies regulate the core activities
- Leaving sufficient room for individual decisions and exception handling

One specific mechanism to support this is the delegation of obligation policies
The Alloy Specification Language

Alloy is a general lightweight modeling language similar to Z and OCL

Tries to bridge the gap between executable and declarative models

Key features:
- Supports object-oriented specification
- Compact syntax and complete semantics
- Precise specification (Syntax/Type checks)
- (Graphical) Analysis facilities based on off-the-shelf satisfiability (SAT) solvers
- Supports incremental specification process
Organisational Obligations

The source of Obligations:
- An organisation’s set of top-level goals, driven by general economic, legal and moral goals
- Identical to the source of authority:
  - both are created by stakeholders and
  - refined down the organisational hierarchy

Why do we delegate obligations?
- Lack of Resources
- Competence
- Specialisation
- Organisational policies

\{ Organisational factors
\}

\{ Policy-based factors

Delegating Policy Objects (1)

Define most general delegation function:
- Covers both authorisations and obligations
- Properties:
  - Delegating subject holds object before delegation
  - Receiving subject holds object after delegation

This will allow for several alternative models to be generated, specifically:
- The delegating subject keeps policy object
- The delegating subject loses policy object
Delegating subject keeps the policy object

next

State_0 → State_1

 Subject_1

Subject_0

holds

PolicyObject_0

Subject_0

Subject_1

holds

holds

PolicyObject_0
Delegating subject loses the policy object
Delegating Policy Objects (2)

- A general delegation model is not sufficient
- Different constraints apply to the delegation of authorisations and obligations

<table>
<thead>
<tr>
<th></th>
<th>Authorisation</th>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Grantor</strong></td>
<td>Must hold before delegation</td>
<td>Must hold before delegation</td>
</tr>
<tr>
<td></td>
<td>Can hold after delegation</td>
<td>Must not hold after delegation</td>
</tr>
<tr>
<td><strong>Grantee</strong></td>
<td>Can hold before delegation</td>
<td>Must not hold before delegation</td>
</tr>
<tr>
<td></td>
<td>Must hold after delegation</td>
<td>Must hold after delegation</td>
</tr>
<tr>
<td><strong>Multiple</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Multiple</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiple</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Obligations require Authority

To perform the actions required in an obligation a subject needs sufficient authority.

Delegation also needs to support this:
- No principal should be delegated an obligation he can not discharge due to a lack of authority.
- No principal should be delegated authority which is not required.

Possible invariants:
- Authorisation-centric
- Obligation-centric
- Well-balanced
Review Policies (1)

- Continual creation, delegation and discharge of obligations causes unstable situations
- Unstable means uncertainty about status of obligation
  - Discharge
  - Effect of discharge
- Accountability is needed
- Review as a policy to ensure that delegated obligations are discharged
- Delegating subject remains accountable
Review Policies (2)

- Review is a post-hoc control for delegated obligations
- A review policy is created as the result of a delegation operation for an obligation
- It is a specific type of obligation
- This is in line with the object model of Ponder
Extended Object Model

PolicyObject

Authorisation

Obligation

Review
Review Policies (3)

- Relationship between actions in obligation and review policies is application dependent.
- Review actions must correspond to the information generated by executing the obligation actions.
- This must have been specified by the application administrator.
Expressing a Review in Alloy (1)

Extending the Obligation signature with two new relations:
- Indicating what action is reviewed by which review action
- Expressing the target of a review policy

Extending delegation function:
- When an obligation is delegated the delegating subject loses its assignment to the obligation
- A review policy (with the delegated obligation as its target) is created and assigned to the delegating subject
Expressing a Review in Alloy (2)
Summary of Review

- A review is an obligation on an obligation
- Required review actions provide application specific information how to perform review
- Creation of a review needs to be supported by the creation of matching authorisations allowing the reviewer to perform the demanded actions
- An obligation cannot be simply delegated:
  - The relevant review actions must have been identified earlier
Review Example: Before State

delegates()

Subject_0
(Subject, Andreas)

Subject_1
(Subject, Jonathan)

obliged_to

Obligation_0
(report_4th_q_Sales, Obligation)

requires

Action_0
(access_report_generator)

requires

Action_1
(access_sales_database)
Review Example: After State
Delegation in Ponder (1)

Ponder is a declarative, object-oriented language for the specification of distributed system policies.

- It provides authorisation (+/-), obligation, refrain and delegation policies.
- Delegation policy specifies authority to delegate, i.e. to execute a delegate() method.
Delegation in Ponder (2)

- Ponder supports the delegation of authorisation policies
- Ponder does not support the obligation to delegate authorisation policies
- Ponder does not support the delegation of obligation policies
Delegation in Ponder (3)

Example:

“A delegates \texttt{print} (action) \texttt{report} (object) to B.”

Requires the following policies:

I. Obligation policy to perform delegation of obligation
II. An authorisation policy for \texttt{A} to create obligation for \texttt{B}
III. P0: oblig \texttt{A}, \texttt{report}, \texttt{print}
IV. An obligation policy to perform delegation of authority
V. P1: auth+ \texttt{A}, \texttt{report}, \texttt{print}
VI. P2: deleg P1, \texttt{A}, \texttt{B}, \texttt{report}, \texttt{print}
Summary & Conclusion

- Suitability of Alloy for modelling and analysing delegation of policies
- Obligations and Authorisations should be balanced
- A distinction needs to be made when delegating authorisations and obligations
- The delegation of obligation policies must cause review policies to be created
- Ponder does not explicitly support authority to delegate obligations or the obligation to delegate authority or obligations
  - Can and should this be integrated?
Future Work

Integration into a control principle framework:

- Relationships between Separation of Duty properties and Delegation controls
- Delegation of review controls
- Supervision controls
- Revocation of delegated controls
URLs

SACMAT presentation:
http://www-users.cs.york.ac.uk/~andreas/SACMAT02/lightweight.ppt

Policy presentation:
http://www-users.cs.york.ac.uk/~andreas/Policy02/delegation.ppt
Example: Before State

- Jon
  - obliged to
    - report 4th q. Sales
      - delegates()
      - requires
        - access report generator ()
        - access sales database ()
- Andrew
Example: After State

Jon obliged to review report 4th q. sales

requires

check logfile ()

view report ()

Andrew obliged to report 4th q. Sales

requires

access report generator ()

access sales database ()

reviewed by

target
The Alloy Specification Language

- Based on first-order logic, similar to Z and parts of the Object Constraint Language (OCL).
- The paper uses “old” Alloy, here I use the “new” version.
- Structured, modular specification using signatures, functions, facts and assertions
- The most important change is the absence of a built in notion of state (pre/post)
- Behaviour can be analysed over arbitrary sequences of states by treating a state as an object (*Objectification of State*)
- Example specification:

```alloy
sig PolicyObject{}
sig Subject{
  holds: set PolicyObject}
fact {
  all s : Subject | #s.holds < 3}
fun somestate{}()
```

![Diagram of Alloy specification example](image)
Objectification of State

This requires:

- **Definition of a state signature**
  
  ```plaintext
  sig Subject{
    holds: set PolicyObject
  }
  
  sig State{
    holds: Subject -> PolicyObject
  }
  ```

- **Definition of a state sequence**
  
  ```plaintext
  static sig State_Sequence {
    disj first, last : State,
    next: (State - last) !->! (State - first)
  }
  
  ...
  
  fact {all s : State | s in s.first.*next}
  
  ...
  ```

<table>
<thead>
<tr>
<th>Subject</th>
<th>Policy Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Pol 1</td>
</tr>
<tr>
<td>B</td>
<td>Pol 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Subject</th>
<th>Policy Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Pol 1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Pol 1</td>
</tr>
</tbody>
</table>
A general delegation model is not sufficient. Different constraints apply to the delegation of authorisations and obligations. To model this we partition the policy object signature:

```
disj sig Authorisation extends PolicyObject{}
disj sig Obligation extends PolicyObject{}
```

These constraints can be summarised as:

<table>
<thead>
<tr>
<th></th>
<th>Authorisation</th>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Grantor</strong></td>
<td>Must hold before delegation Can hold after delegation</td>
<td>Must hold before delegation Must not hold after delegation</td>
</tr>
<tr>
<td><strong>Grantee</strong></td>
<td>Can hold before delegation Must hold after delegation</td>
<td>Must not hold before delegation Must hold after delegation*</td>
</tr>
<tr>
<td><strong>Multiple</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td>Downwards Upwards Horizontal</td>
<td>Downwards Upwards Horizontal</td>
</tr>
</tbody>
</table>
A General Delegation Function

The most general delegation function is defined as follows:

```
fun delegation_1 (disj s, s' : State, disj subj1, subj2 : Subject,
                  pol_obj : PolicyObject) {
  (subj1 -> pol_obj) in s.holds
  (subj2 -> pol_obj) in s'.holds
}
```

Constrain state transitions to follow this function

```
all states : State - last | some s = states | some s' = states.next |
  some disj subj1,subj2 : Subject |
  some pol_obj : PolicyObject |
  delegation_1(s,s',subj1,subj2,pol_obj)
```

This will allow for several models to be generated, specifically:

- The delegating subject keeps policy object
- The delegating subject loses policy object
The delegating subject keeps the policy object
The delegating subject loses the policy object
An extended delegation operation needs to cater for the delegation of authorisations and obligations

fun delegation_2 (disj s, s' : State,
    disj subj1, subj2 : Subject,
    pol_obj : PolicyObject)
{
    pol_obj in Authorisation => (subj1 -> pol_obj) in s.holds
        (subj1 -> pol_obj) in s'.holds
        (subj2 -> pol_obj) in s'.holds

    pol_obj in Obligation => (subj1 -> pol_obj) in s.holds
        (subj1 -> pol_obj) !in s'.holds
        (subj2 -> pol_obj) in s'.holds
}
Obligations require Authority

To perform the actions required in an obligation a subject needs sufficient authority.

A new action signature

\[
\text{sig Action}
\]

Two new relations to be added to the state signature

\[
\text{requires: Obligation} \rightarrow \text{Action}
\]

\[
\text{permits: Authorisation} \rightarrow \text{Action}
\]
Balancing Obligations and Authorisations

Possible invariants
- Authorisation-centric
- Obligation-centric
- Well-balanced

Define two functions to obtain
- authorised actions for a subject in a state
  ```
  fun ret_auth_action (subj : Subject, s : State): set Action{
    result = {a : Action | a in (subj.(s.holds)).(s.permits)}
  }
  ```
- required actions for a subject in a state
  ```
  fun ret_req_action (subj : Subject, s : State): set Action{
    result = {a : Action | a in (subj.(s.holds)).(s.requires)}
  }
  ```

Example for authorisation centric
```
fact {all subj : Subject | all s : State | ret_auth_action(subj,s) in ret_req_action (subj, s)}
```
Expressing a Review in Alloy (1)

- Extending the Obligation signature
  \[\text{disj sig Review extends Obligation}\{\}\]

- A new relation indicating what action is reviewed by which review action
  \[\text{sig Action}\{\}
   \text{reviewed\_by: Action}\}

- A new relation expressing the target of a review policy
  \[\text{disj sig Review extends Obligation}\{\}
   \text{target: Obligation}\}

- Relationship between actions in obligation and review policies is application dependent
Expressing a Review in Alloy (2)

- Expand delegation function
- When an obligation is delegated the delegating subject loses its assignment to the obligation
- A review policy (with the delegated obligation as its target) is created and assigned to the delegating subject

```alloy
fun review_delegation_1 (disj subj1, sub2 : Subject, ...){
  ...
  pol_obj in Obligation =>
    some rev : Review | no (s.holds).rev &&
    no rev.(s.target) &&
    (rev -> pol_obj) in s'.target &&
    (subj1 -> pol_obj) in s.holds &&
    (subj1 -> pol_obj) !in s'.holds &&
    (subj2 -> pol_obj) !in s.holds &&
    (subj2 -> pol_obj) in s'.holds &&
    (subj1 -> rev) in s'.holds &&
    frame (s',s,rev, pol_obj) ...
...
```
Before Delegation

Jon delegates to Andy

Jon holds

report 4th q. Sales

access report generator ()

access sales database ()

Andy holds

report 4th q. Sales

After Delegation

Jon holds

review report 4th q. sales

requires

check logfile ()

view report ()

requires

access report generator ()

access sales database ()

Andy holds

report 4th q. Sales

target

reviewed by

access report generator ()

access sales database ()