Usage Control for Collaborative Systems

Alvaro Arenas

E-Science Centre
STFC Rutherford Appleton Laboratory
Collaborative Systems

- Provide scalable access to distributed computing facilities

- Grid – a global network of computers that can operate as one vast resource
Grids

- Resource sharing
  - Computers, storage, sensors, networks, telescopes …

- Coordinated problem solving
  - Beyond client-server: distributed data analysis, computation, collaboration, …

- Dynamic, multi-institutional Virtual Organisations
Large Hadron Collider

Downtown Geneva

Mont Blanc (4810 m)

Concorde (15 Km)

CD stack with 1 year LHC data! (~20 Km)

Balloon (30 Km)

Mt. Blanc (4.8 Km)
A Cure for Malaria?

WISDOM Project – Grid-enabled software to screen molecules for their ability to disable a crucial malaria protein
How Secure are Collaborative Systems?

- There are some security risks due to opening our systems to other systems/users

- If the other users/systems are not trusted, there is need for mechanisms to mitigate such risks
Usage Control Model

- Encompass traditional access control, trust management and digital right management

- Usage control based on
  - Authorisations
  - Obligations
  - Conditions
  - Mutability of attributes
  - Continuity of decision

Subjects and Objects

- **Subjects**: entities that perform actions on Objects
  - Characterised by attributes:
    - Identity; Role; Reputation; Credits; ...

- **Objects**: entities that are used by Subjects.
  - Characterised by attributes:
    - Value; Role permission; …
Mutability of Attributes

• Attributes of subjects and objects
  – Can be static (IMMUTABLE)
  – Can be updated (MUTABLE)
    • Before usage (PRE)
    • During usage (ONGOING)
    • After usage (POST)

• Example
  – A storage service charges its users when they read documents. The credit attribute of an user is updated before he reads a document
Authorisations

- Predicates for usage decisions that evaluate
  - Subject Attributes
  - Object Attributes
  - Right/Actions

- Example: a computational service exploits a security policy to decide whether the user U can perform the action “read” on the file “a.txt”
Obligations

- Predicates stating mandatory requirements to be performed by the Subject
  - Actions
  - ...

- Example: the user of a storage service must download the license agreement before downloading any other document.
Conditions

- Environmental or system based decision factors
  - Not directly related with Subjects and Objects
  - e.g.
    • Current local time
    • Current system workload
    • System status

- Example: night-users can submit jobs to a computational resource only from 8pm to 8am
Usage Right Evaluation
• The evaluation of usage right can be performed
  – Before the usage (PRE)
  – During the usage (ONGOING)
    • The right could be revoked and the action interrupted
    • Used for long lived actions (days, months,..)
Formal Models for UCON

- **TLA** – original formal model, used to characterise the different submodels in UCON (preA, onA, preB, onB, etc)

- **Process Algebra** – The PolPA language. More operational than original proposal, suitable for design/implementation

- **Interval Temporal Logic** – In the middle of above proposals, exploit the existence of sequential operator in ITL
A Process-Algebra Policy Language for UCON

- **POLPA: POLicy Process Algebra**
  - Developed at ITT-CNR, Italy (Martinelli et al)

- Operational language based on process description languages
  - Describe the allowed sequential behaviours of actions
  - CSP + variables

- Policies can thus be formally compared, minimized, verified, ...

- All UCON models can be encoded

POLPA: POLicy Process Algebra

· Syntax

\[ P ::= \text{stop} \parallel \text{skip} \parallel \alpha(\vec{x}).P \parallel p(\vec{x}).P \parallel \vec{x} := \vec{e}.P \]

\[ \parallel P_1 \text{ or } P_2 \parallel P_1 \text{ par} \{\alpha_1, \ldots, \alpha_n\} P_2 \parallel Z \]

\(\alpha\) actions, \(p\) predicates, \(x\) variables, \(e\) expressions (values)

(actions and predicates possibly contain variables)
Operational Semantics

- Behaviour is modeled through a *labeled transition system*

\[
\langle \mathcal{P}, \text{Act}, \{ \xrightarrow{\alpha} \}_\alpha \in \text{Act} \rangle
\]

- ternary relation representing a transition relation between processes through an action. Transition happens according to rules as:

\[
\frac{\text{premise}}{\text{conclusion}} [\text{side condition}]
\]
(prefix) \[
\alpha.P \xrightarrow{\alpha} P
\]

(or) \[
P \xrightarrow{\alpha} P'
\]

(pred) \[
P(\vec{e}/\vec{x}) \xrightarrow{\alpha} P'
\]
\[
p(\vec{e}/\vec{x}).P(\vec{e}/\vec{x}) \xrightarrow{\alpha} P' \quad [p(\vec{e}/\vec{x}) = \text{true}]
\]

(par\textsubscript{1}) \[
P \xrightarrow{\beta} P'
\]
\[
P\par\{\alpha\} Q \xrightarrow{\beta} P'\par\{\alpha\} Q \quad [\beta \notin \{\alpha_1, \ldots, \alpha_n\}]
\]

(par\textsubscript{2}) \[
\]
\[
P \xrightarrow{\beta} P' \quad Q \xrightarrow{\beta} Q'
\]
\[
P\par\{\alpha\} Q \xrightarrow{\beta} P'\par\{\alpha\} Q' \quad [\beta \in \{\alpha\}]
\]

(const) \[
P \xrightarrow{\alpha} P'
\]
\[
Z \xrightarrow{\alpha} P' \quad [Z = \text{def} P]
\]
Actions and Predicates

- Communication
- Scenario-specific actions like Write/Read/Copy/Print...
  \[ \alpha(x) := \alpha_{com.m}(x) || \alpha_{spe.c}(x) || \ldots \]

- \( p \) are conditions that must be satisfied to go ahead

  \begin{align*}
  p(x) &:= \\
  &\begin{cases} 
  \bar{x} = \bar{e}, & (\bar{x} \leq \bar{e}), \ldots \\
  p_{time}(x) & \text{Time-based predicates} \\
  p_{loc}(x) & \text{Location-based predicates} \\
  p_{rol}(x) & \text{Role-based predicates} \\
  p_{tru}(x) & \text{Trust-based predicates} \\
  \neg p(x) & \text{Negation} \\
  p(x) \land p(x) & \text{Conjunction} \\
  p(x) \lor p(x) & \text{Disjunction} \\
  ( p(x) ) & \text{Precedence}
  \end{cases}
  \end{align*}
Example – UCON PreA

- Authorization is performed before the right is exercised
  - With immutable Attributes
    - traditional authorization
  - With pre update of Attributes
    - Attributes value is updated before the usage is started

- Pay per use with pre-paid credit
  - Authorizations granted when: \( credit(s) > value(o,r) \)
  - preUpdate(credit(s)): \( credit(s) = credit(s) - value(o,r) \)
Example – UCON PreA

• With post-update of attributes
  – Attributes value is updated after the usage is terminated

• Membership based payment
  – Authorisation granted when: clubmember(s)
  – postUpdate(exp(s)): exp(s) += value(o,r) x utime(s,o)
Usage Control in Grids

- Security and Usage Requirements for Grids
  - Goal-oriented requirements engineering
  - From requirements to specifications

- Specification and Design of Grid-Based Systems
  - Modelling policy-based systems in Event-B
  - Obligations in Event-B
  - Refinement of UCON policies
  - Usage control and reputation

- Architecture and Implementation of Usage Control for Grids
  - Reference monitor at both collaboration level (global / virtual organisation level) and node level
  - XACML and POLPA
Security and Usage Requirements for Grids

- Extending the KAOS requirements
  - Taking into consideration the (Grid) middleware
    - Adding a VO Model as a middleware abstraction
  - Modelling trust relationships by adding a Trust Model
    - Ownership, capability and trust
    - Delegation
    - Distrust and monitoring
  - Defining requirement patterns for typical usage requirements
  - Tool support

KAOS (Knowledge Acquisition in autOmated Specifications) is a formal requirement engineering methodology consisting of several models including:

- The Goal Model
- The Operation Model
- The Object Model
- The Agent Model
- The Anti-Goal Model
Our Scenario: Grid-based Content Management System

- VO for generating a complex digital product
- VO has defined number of member organisations
- Product generation process is knowledge/content-intensive
- Product generation process is structured as a workflow
- Policies may be applied to control usage/access to resources along the workflow
The Goal/Operation Models
Verifying Goal Refinement

• The refinement must satisfy the following assertions

\[ G_1, \ldots, G_n, D \models G \]  (completeness)

\[ \bigwedge_{j \neq i} G_j, D \not\models G \text{ for each } i \in [1..n] \]  (minimality)

\[ G_1, \ldots, G_n, D \not\models false \]  (consistency)

• Using Alloy to check these assertions
Ownership, Capability and Trust

In our GCM system scenario:

**Trust**

- **Trustor**: Editor
- **Goal**: [ContentReviewed]
- **Trustee**: Reviewer
- **Description**: Agent Editor trusts Reviewer to enforce goal [ContentReviewed]
Conflict of Interest Pattern

Goal AvoidConflictOfInterest
FormalDef \((\forall u:User; r, r':Resource) \textit{noConflict}(u, r, r')\)

Goal AccessAuthorizedWithinSameCompany
FormalDef \((\forall r:Resource; \exists u:User; \exists r':Resource)\)
\[
\text{hasAccessed}(u, r) \land \text{sameOrganisation}(r, r') \Rightarrow \neg(\text{hasAccessed}(u, r'))
\]

Goal AccessAuthorizedWithinOtherConflictSet
FormalDef \((\forall r:Resource; \exists u:User; \exists r':Resource)\)
\[
\text{hasAccessed}(u, r) \land \text{differentConflictSet}(r, r') \Rightarrow \neg(\text{hasAccessed}(u, r'))
\]

Goal ChineseWallAuthorizedCases
FormalDef \((\forall u:User; r, r':Resource)\)
\[
\text{hasAccessed}(u, r) \Rightarrow (\text{sameOrganisation}(r, r') \lor \text{differentConflictSet}(r, r'))
\]
Modelling Virtual Organisations in Event-B

- Model the VO lifecycle in Event-B
- Exploit Event-B refinement methodology to include the VO layers of abstractions

Modelling Obligations in Event-B

- Introduce triggers to constraint order of events
  - Triggers are dual to Guards
    - When the guard is not true, the event must not occur
    - When the trigger is true, the event must occur

- Interpret triggers as a syntactic sugar
  - Triggers model obligations by constraining when other events are permitted

A Mine Sump

- Water seeps into the sump irregularly
  - pump used to keep level between bounds
- Methane can accumulate in the mine
  - alarm sounded when methane detected

Requirements

- pump must be activated when high water is sensed (risk of flooding)
- pump must be deactivated when low water sensed (risk damage to pump)
- alarm must be sounded immediately methane is detected (evacuation of personnel)
- pump must be deactivated when methane detected (risk of explosion)
INVARIANTS

lowwater : Bool
highwater : Bool
methane : Bool
pump : {ON, OFF}
bell : {ON, OFF}

EVENTS

high_water_detected
WHEN highwater = true THEN pump := ON END

low_water_detected
WHEN lowwater = true THEN pump := OFF END

methane_detected
WHEN methane = true THEN bell := ON || pump := OFF END

What about Fairness?
Is Fairness Enough?
Need explicit obligation

Modelled by prohibition
AND not(methane = true)
AND not(methane = true)
Triggered Events - NEXT

EVENT \( f \) WHEN \( T \) NEXT \( R \)
- forces event \( f \) to be the next one to be executed
  - when condition \( T \) is true

Represents the obligation \( \square(\neg T \Rightarrow \circ f) \)
- where symbol \( \circ \) denote the next temporal operator

Modelled by prohibiting all other events
- extending the guard of all other events with the negation of \( T \)

EVENT \( e_i \) WHEN \( G_i \) THEN \( S_i \) END
EVENT \( f \) WHEN \( T \) NEXT \( R \) END
EVENT \( e_i \) WHEN \( G_i \land \neg T \) THEN \( S_i \) END
EVENT \( f \) WHEN \( T \) THEN \( R \) END
INSENSITIVES

lowwater : Bool
highwater : Bool
methane : Bool
pump : {ON, OFF}
bell : {ON, OFF}

EVENTS

high_water_detected
  WHEN highwater = true AND not(methane = true)
  THEN pump := ON
END

low_water_detected
  WHEN lowwater = true AND not (methane = true)
  THEN pump := OFF
END

methane_detected
  WHEN methane = true
  NEXT bell := ON || pump := OFF
END

THEN replaced with NEXT ...

... as shorthand for these
• Bounded eventuality
  ➢ EVENT $f$ WHEN $T$ WITHIN $n$ NEXT $R$

  “if trigger $T$ becomes true and remains true for $n$ events
  then event $f$ must be executed within these $n$ events”

  (If trigger $T$ becomes false within these $n$ steps
  then the obligation is cancelled)

• Represents the obligation $\Box \Diamond_{\leq n} (T \Rightarrow f)$
  ➢ a “queuing” version of the “leads-to” modality
    • i.e. $T$ and not $f$ cannot be sustained for more than $n$ events

• Standard “leads-to”, $\Box (T \Rightarrow \Diamond_{\leq n} f)$, can also be modelled.
WITHIN Events

- Modelled by extending the state with a counter for each WITHIN
  - For each WITHIN event \( f \), we add counter \( \text{counter}_f \) which
    - is set to \( n \) whenever \( T \) is false
    - is decremented each time another event is executed while \( T \) is true
    - prohibits other events if it reaches 1

EVENT \( e \) WHEN \( G \) THEN \( S \) END
EVENT \( f \) WHEN \( T \) WITHIN \( n \) NEXT \( R \) END

INVARIANT \( \ldots \quad \Box \ 1 \leq \text{counter}_f \leq n \)
INIT \( \ldots \quad \| \quad \text{counter}_f := n \)
EVENT \( e \) WHEN \( G \quad \Box \quad (\neg T \lor \text{counter}_f > 1) \) THEN
\( S \quad \| \quad \text{if } T \ \text{THEN } \text{counter}_f := \text{counter}_f -1 \ \text{ELSE } \text{counter}_f := n \) END
EVENT \( f \) WHEN \( T \) THEN \( R \quad \| \quad \text{counter}_f := n \) END

Remove this for usual leads-to
From Requirements to Specifications

- Linking KAOS requirements with Event-B specifications
  - Event-B with obligations needed since KAOS models a maximum set of permitted behaviours as well as a minimum set of obliged ones

- Use model-checking techniques to verify to an Event-B machine is a model of a set of KAOS requirements
  - Using the ProB animator and model-checker
Table 1. Patterns for KAOS Goals

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Formal Definition</th>
<th>Event-B Operationalisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate Behaviour</td>
<td>$C \Rightarrow \circ S$</td>
<td>$\text{EVENT } e \text{ WHEN } \overline{C} \text{ NEXT } \overline{S} \text{ END}$</td>
</tr>
<tr>
<td>Bounded Behaviour</td>
<td>$C \Rightarrow \Diamond_{\leq d} S$</td>
<td>$\text{EVENT } e \text{ WHEN } \overline{C} \text{ WITHIN } d \text{ NEXT } \overline{S} \text{ END}$</td>
</tr>
<tr>
<td>Eventually Behaviour</td>
<td>$C \Rightarrow \Diamond S$</td>
<td>$\text{EVENT } e \text{ WHEN } \overline{C} \text{ EVENTUALLY } \overline{S} \text{ END}$</td>
</tr>
<tr>
<td>Restriction Behaviour</td>
<td>$C \Rightarrow S$</td>
<td>$\text{EVENT } e \text{ WHEN } \overline{C} \text{ THEN } \overline{S} \text{ END}$</td>
</tr>
</tbody>
</table>

Table 2. Patterns for Operationalising KAOS Goals with Event-B
Policy Refinement (Decomposition)

- Policy refinement refers to the process of decomposing a global policy (applicable to several nodes) into a set of policies
  - Each policy is applicable in a node
  - It does not refer to the removal of non-determinism (as it is the case of FM refinement)

- Why refinement of UCON policies for Grids
  - It may not always be possible to enforce VO policies at the VO level
  - Therefore, we need to translate and enforce VO policies at the low level of resources
Policy Refinement as Model Transformation

- Following Chadwick’s work, we assume there exists a hierarchy of resources
- Use the ATLAS Transformation Language from INRIA to transform policies in one level (model) into the corresponding lower level
The Transformation Process

- Processes are mapped to their corresponding command sets
- VO resources are mapped to computational resource sets
- VO user is mapped to a set of user ids
- VO environment is mapped to resource-level environment
Transforming POLPA Policies

**VO Policy**

\[
\text{requestaccess}(\text{Bart Simpson, Edit, UK Map}) \cdot \text{permitaccess}(\text{Bart Simpson, Edit, UK Map}). \quad \text{(endaccess(\text{Bart Simpson, Edit, UK Map}).update(\text{edit count var} \text{var}++) +} \quad \neg \text{P.revokeaccess}(\text{Bart Simpson, Edit, UK Map}))
\]

where,

\[
\text{Pdef} = ((\text{resource-id-qualifier} = \text{National Geographic}) \quad \text{AND} \quad (13:00:00 \leq \text{subject:request-time} \leq 14:00:00) \quad \text{AND} \quad (\text{environment:current-day} = \text{weekday}) \quad \text{AND} \quad (\text{action:edit count var} \leq 3) \quad \text{AND} \quad (\text{environment:status flag} = \text{edit}))
\]

**Resource Policy**

\[
!\left( \text{requestaccess}(\text{abc1234, read, UK Map/Topography Data Set}) \cdot \text{permitaccess}(\text{abc1234, read, UK Map/Topography Data Set}). \quad \text{(endaccess(\text{abc1234, read, UK Map/Topography Data Set}).update(\text{edit command count var} \text{var}++) +} \quad \neg \text{P.revokeaccess}(\text{abc1234, read, UK Map/Topography Data Set}))
\]

where,

\[
\text{Pdef} = ((\text{resource-id-qualifier} = \text{National Geographic}) \quad \text{AND} \quad (13:00:00 \leq \text{subject:request-time} \leq 14:00:00) \quad \text{AND} \quad (\text{environment:current-day} = \text{weekday}) \quad \text{AND} \quad (\text{action:edit command count var} \leq 3) \quad \text{AND} \quad (\text{environment:status flag} = \text{edit}))
\]
Implementing UCON for Grids: Some Issues

- **Attribute Management**
  - How to represent, store, retrieve, update attributes

- **Ongoing Controls**
  - Usage revocation
  - Main novelty w.r.t. usual access control

- **Conditions**
  - Environmental conditions for Grids

Actions in a Usage Process

- Initial state: tryaccess
- Requesting: permitaccess
- Accessing: onupdate, preupdate, endaccess, postupdate
- Denied and preupdate
- Revokeaccess and postupdate
- End

Diagrams illustrating the flow of actions in a usage process.
• We assume the following usage control actions

  – tryaccess(s, o, r): performed by subject s when performing a new access request (s, o, r)
  – permitaccess(s, o, r): performed by the system when granting the access request (s, o, r)
  – revokeaccess(s, o, r): performed by the system when revoking an ongoing access (s, o, r)
  – endaccess(s, o, r): performed by a subject s when ending an access (s, o, r)
  – update(attribute): updating a subject or an object attribute.
Architecture

Grid Node

PEP

condition manager

attribute manager

obligation manager

PDP

policy

tryaccess(s,o,r)

get(c)

get(a)/update(a,v)

get(o)

permitaccess(s,o,r)/denyaccess(s,o,r)

denoaccess(s,o,r)

revokeaccess(s,o,r)

value

value
Future Work

• Modelling richer policy-based system
  – Usage control in business processes
  – Usage control in Grid-based operating systems
  – Data-centric security
  – Service-oriented computing, clouds, …

• More on obligations

• More on policy analysis
  – Policy containment, conflict analysis, …
  – Formal models for policy decomposition
Conclusion

- Usage control has shown to be a good paradigm for mitigating security risk in collaborations

- Reasoning about trust and security at all levels
  - Vertical approach, from requirements down to specification, design and implementation
Acknowledgement

Several people have contributed to the work presented here, in particular:

- STFC, UK: Juan Bicarregui, Benjamin Aziz, Brian Matthews
- CETIC, Belgium: Philippe Massonet, Christophe Ponsard, Gautier Dallons
- CNR, Italy: Fabio Martinelli, Paolo Mori, Marinella Petrocchi
- Interplay, Italy: Giovanni Cortese
Usage Control for Collaborative Systems

Alvaro Arenas

E-Science Centre
STFC Rutherford Appleton Laboratory