Obligation Monitoring in Policy Management

Claudio Bettini*,
Sushil Jajodia#,
X. Sean Wang#,
Duminda Wijesekera#

Università di Milano, Italy* and George Mason University#
Need for Provisions and Obligations

- Policies are widely used in many different systems
- Yes/no response to every request is just not enough
- **Provisions**: Conditions to be satisfied before permission is considered
- **Obligations**: Conditions to be fulfilled as a consequence of “yes”
Eg: Electronic Loan Application

• Provisions: Registered account holder
  – Either already registered or register now!
  – Some actions need to be taken in order to satisfy the obligations

• Conditions:
  – Have a good credit history
  – Makes enough money to pay back

• Then, the bank sells the loan
Example continued

• Obligations
  – Customer needs to make up her mind in a week
  – Agree to abide by following conditions
  1. Have to pay an installment every month by the due date
  2. If not, have to pay installment+surcharge within two weeks grace period
  3. Failing (2), the loan will be cancelled and property re-processed.
Representation of the Problem

• Access control uses a rule-based system.
• Rules are Horn clauses decorated with provisions and obligations.
• Provisions and obligations are constructed from conjunctions and disjunctions of literals.
• Have disjoint sets of atoms for predicates in rules and their decorations
• Terms are made of constants and variables shared across both kinds of predicates.
Example Specification

1. canDo(customer, loan, read) <-
   Prov: register(customer)
2. canDo(customer, loan, apply) <-
   canDo(customer, loan, read),
   Prov: signedLetterOfIntent(customer, loan)
Example Continued

3. access(customer,loan,selfApprove) <-
   reliable(customer,score,time), score>7.2,
   computePayment(customer,loan,amount),
   income(customer,salary), salary>2.amount

Obl 1: signWithin(customer,loan,time,7)

Obl 2: payOffLoan(customer,loan,time+3600)
Selecting Among Options

\[
\begin{align*}
\text{register}(\text{cus}, \text{regular}) & \leftarrow \\
& \quad \text{Prov: pay}($10) \\
\text{register}(\text{cus}, \text{preferred}) & \leftarrow \\
& \quad \text{Prov: pay}($30) \\
\text{canDo}(\text{cus}, \text{purchase}, x, \text{price}) & \leftarrow \text{register}(\text{cus}, \text{regular}) \\
& \quad \text{Prov: pay}($\text{price}) \\
\text{canDo}(\text{cus}, \text{purchase}, x, \text{price}) & \leftarrow \text{register}(\text{cus}, \text{preferred}) \\
& \quad \text{Prov: pay}($\text{(80\%).price}) \\
\end{align*}
\]
Optimization

• Can compute cost for each option, and optimize the purchasing plan.
• Similarly, can have optimizations over obligations
• Paper in VLDB details some optimization strategies.
Structure of Obligations-I

• Every Obligation has:
  – A fulfilling clause
    • A finite set of actions to be taken by the system when the obligation is fulfilled
  – A defaulting clause: Compensating Activity
    • An instance of an obligation
    • A finite set of obligations to be taken by the system when the obligation is defaulted

• Constraint: All obligation chains are finite and no cycles.

• Action: sending or receiving of an event
Compensations

- Use a non-negative real number for scaling reliability of the customer.
  - Very much like the credit rating
- Modeled by a predicate
  `reliable(subject, score, time)`
- The system updates the reliability by sending the event
  `send(adjReliable, sys, time, sub, score)`
Example

OBL Definition:
\text{payByDate}(\text{cust, loan, t, pmnt, penalty, upScore, dwnScore})

FUL:[\text{Action List:}]
\{\text{send(ackReciept, cust, now, loan, pmnt),}
\text{send(adjRel, system, cust, now, upScore)}\}\]

DEF:
\text{OBL: payByExtDate}(\text{cust, time, pmnt+penalty})

[\text{Action List:} \text{send(reminder, cust, now, loan, pay-penalty),}
\text{send(adjRel, system, cust, now, -downScore)}]
Structure of Obligations - II

Obligations are hierarchically as

• Using one obligation in the defining clause of another

• If $O_1, O_2$ are obligations then so are $O_1 \land O_2, O_1 \lor O_2$

• If $O(x)$ is an obligation definition template then $[\text{for } x=1 \text{ to } n \ O(x)]$ is an obligation

• If $p$ is a predicate then $[\text{if } p \text{ then } O]$ is an obligation
Example

\begin{equation}
\text{buyWithin}(\text{cust}, \text{loan}, \text{time}, 7) \land \\
[ \text{for } n=1 \text{ to } 36 \\
\quad \text{if } (\text{not } (\text{receive}(\text{loanCancelNotice}, \text{cust}, \text{loan}) \land \\
\quad \quad (t < 30n)) \{ \\
\quad \quad \quad \text{payByDate}(\text{cust}, 30n+5, \text{pay}) \lor \\
\quad \quad \quad \text{payByExtDate}(\text{cust}, 30n+15, \text{pay}+100) \\
\quad \quad \}
\}
\end{equation}
Monitoring Obligation Fulfilling

• Recursively build an algorithm to monitor obligations, based on the system having an event service.

• Note: For loops are syntactic sugar for conjunctions of obligations. That is for $i=1$ to $n$ $O(i)$ is equivalent to $O(1) \land O(2) \ldots O(n)$
Basic Step

• For each atomic obligation $O$ let
  
  – $\text{fulAct}(O)$ be the set of actions corresponding to the fulfillment of $O$.

  The obligation is met iff the “system” receives all events in $\text{fulAct}(O)$.

  In that case the system must generate all events listed in the fulfillment clause of the obligation.

  Else generate all actions listed in the DEF clause.
Conditional Step

• If the obligation $O$ is of the form
  
  \texttt{if condition \textit{O'}}

• Change all \texttt{receive} predicates in the condition to send predicates.

• If events corresponding to the transformed predicate have been sent by the system then fulAct($O$) is fulAct($O'$)
Conjunctions and Disjunctions

- \( \text{fulAct}(O \land O') = \text{fulAct}(O) \cup \text{fulAct}(O') \)  
  - That is \( O \land O' \) is fulfilled iff both \( O, O' \) are fulfilled

- \( \text{fulAct}(O \lor O') = \text{fulAct}(O) \cap \text{fulAct}(O') \)  
  - That is \( O \lor O' \) is fulfilled iff either of \( O, O' \) are fulfilled.
Related Work - I

• PONDER has event triggered condition action rules as obligation policies.

• Provisional authorizations have been proposed by Jajodia et al.
  – Computes weakest precondition under which a provisional authorization can be granted

• Incorporated into XML by Kudo et al, where a provision models
  – Verifying, encrypting, transforming text etc.
Related Work - II

• Chomicki et al. proposed using a past time temporal logic for specifying constraints in policies. Obligations can be seen as a kind of constraint placed on the future time.

• Minsky et al. First applied obligations to security policies and coordination, in which obligations are strictly enforced and not only monitored.

• Work of Donson et al.
Work in Progress

• Reasoning about provisions and obligations
• Translating provisions and obligations into logic programs without events by using event calculus ala Kowalski et al.
• Monitoring obligations that have complex quantitative temporal conditions