Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

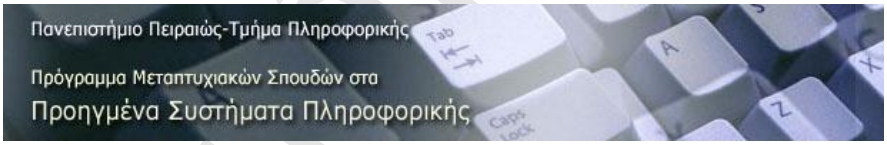
Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη χρήση Λογισμού Γεγονότων (Event Calculus)**
Ονοματεπώνυμο Φοιτητή	**Αδαμόπουλος Γεώργιος του Θεοδώρου**
Αριθμός Μητρώου	**ΜΠΣΠ 07034**
Κατεύθυνση	**Δίκτυα**
Επιβλέπων	**Χρήστος Δουληγέρης, Καθηγητής**
Συνεπιβλέπων	**Δρ. Σαράντης Μητρόπουλος**

Μεταπτυχιακή Διατριβή Γεώργιος Αδαμόπουλος



Ημερομηνία Παράδοσης **Μήνας Έτος**

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Χρήστος Δουληγέρης Καθηγητής	Δημήτριος Βέργαδος Λέκτορας	Παναγιώτης Κοτζανικολάου Λέκτορας

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Μεταπτυχιακή Διατριβή

Γεώργιος Αδαμόπουλος

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Μεταπτυχιακή Διατριβή Γεώργιος Αδαμόπουλος

Στο Χρήστο μου που είναι μαχητής.

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά θα ήθελα να ευχαριστήσω τον φίλο μου Αλέκο Βαβούση που με (ξανα)έβαλε στη διαδικασία απόκτησης ενός μεταπτυχιακού τίτλου. Τον Δρ. Σαράντη Μητρόπουλο γιατί έφερε στον ορίζοντά μου μια κατηγορία προβλημάτων εξαιρετικού θεωρητικού και πρακτικού ενδιαφέροντος, τόσο για τον security policy manager, όσο και για τον system administrator. Γιατί μου έδειξε ένα χώρο στον οποίο θέλω να είμαι πραγματικά καλός. Και γιατί όποτε χανόμουν στις πολυάριθμες ενδιαφέρουσες περιπτώσεις που άνοιξε στο δρόμο μου η Event Calculus, καθώς μελετούσα το πρόβλημα, με επανέφερε σε μια κεντρική οδό.

Τον προϊστάμενό μου Νίκο Παναγιωτόπουλο γιατί χωρίς αυτόν ο εργοδότης μου δε θα μου χορηγούσε εκπαιδευτική άδεια. Τον τότε Πρόεδρο του εργοδότη μου γιατί με την αρνητική του στάση έδωσε ακόμα μεγαλύτερη αξία σε αυτή μου την προσπάθεια. Τον φίλο και συνάδελφο Γιώργο Κωτσόλη: I kept walking.

Περισσότερο από κάθε άλλον την γυναίκα μου Ειρήνη Μανώλη που μου στάθηκε σε όλη τη διαδικασία και ανέχτηκε τις παραξενιές και τα άγχη μου. Τα παιδιά μου: Τον Θοδωρή που είχε γεννηθεί πριν ξεκινήσω αυτή την προσπάθεια και τα δίδυμα Χρήστο και Κασσάνδρα που ήρθαν λίγο μετά. Και στους τέσσερις δεν αφιέρωσα το χρόνο που θα έπρεπε και τώρα θα επανορθώσω.

ΠΕΡΙΛΗΨΗ

Προσωπικοί υπολογιστές, πληροφορικά συστήματα, εταιρικά και οικιακά δίκτυα απλών και πολύπλοκων διαμορφώσεων ακόμα και δίκτυα Κρίσιμων Υποδομών διασυνδέονται στο Δίκτυο. Από την σύνδεσή τους αυτή απορρέει η ανάγκη για κανόνες πρόσβασης και προστασίας των συστημάτων, ανάγκη αυξημένη σε σχέση με αυτόνομα αποκομμένα συστήματα. Οι κανόνες πρόσβασης περιγράφονται από πολιτικές ασφαλείας, οι οποίες περιγράφονται με ποικίλη πολυπλοκότητα, με υποθέσεις που εννοούνται ή όχι και φυσικά με πολλές εξαιρέσεις. Σε αυτό το περιβάλλον οι διαχειριστές θα πρέπει να υλοποιήσουν, συντηρήσουν και να είναι σε θέση να ελέγχουν την ακεραιότητα των εφαρμοζόμενων πολιτικών σε σχέση με τα περιγραφόμενα και όσα περιμένει ο χρήστης από αυτά. Η αγορά έχει απαντήσει στο κενό που υπάρχει μεταξύ της πολιτικής ασφάλειας και της υλοποίησής της στον εξοπλισμό με συστήματα ad-hoc που όπως εξελίσσονται κατά τη διάρκεια του χρόνου τείνουν να επαληθεύουν τον εμπειρικό 10ο κανόνα του Greenspun για το software engineering («Κάθε αρκετά περίπλοκο πρόγραμμα σε C ή FORTRAN περιέχει μια ανεπίσημη, γεμάτη λάθη, αργή υλοποίηση της μισής Common Lisp»). Η μεταπτυχιακή αυτή διατριβή έρχεται να καλύψει αυτό το γεγονός χρησιμοποιώντας Μαθηματική Λογική και τη γλώσσα PROLOG από την αρχή ώστε να είναι ευκολότερη η περιγραφή, η υλοποίηση, η συντήρηση και ο έλεγχος των εφαρμοζόμενων πολιτικών ασφαλείας. Μέσω του θεωρητικού αυτού υπόβαθρου στόχος είναι η δημιουργία ισχυρών εργαλείων που θα προσφέρουν πραγματική βοήθεια στον διαχειριστή και τον υπεύθυνο ασφαλείας στην καθημερινή του εργασία.

ABSTRACT

Personal computers, information systems, corporate and home networks of either simple or complex configurations and even Critical Infrastructure networks interconnect through the Net. This connectivity defines the need for access control to and protection of the relevant systems, a need which is increasingly pressing especially when compared with isolated systems and networks. Access control rules are described via security policies which vary from simple to complex where assumptions exist regardless of whether they are obvious to users and administrators and quite often with a lot of exceptions. In such an environment the system administrators are expected to deploy, manage, and verify the integrity of what is applied to the equipment compared to the policy itself. The market has responded to the void created by such a need by providing ad-hoc tools that while evolving as time passes tend to prove Greenpsun's empirical 10th law of software engineering ("Any sufficiently complicated C or FORTRAN program, contains an ad-hoc, informally specified, bug-ridden, slow implementation of half of Common Lisp"). This dissertation aims by using Mathematical Logic tools and PROLOG right from the start to provide for easier deployment, management and verification of security policies that should be in place. Via using this theoretical background powerful tools that can offer solid help to the daily tasks of the system administrator and the security manager are proposed.

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Εισαγωγή

Το Διαδίκτυο αποτελεί το κοινό σημείο συνάντησης μιας πληθώρας χρηστών, υπολογιστών και οργανισμών. Οι διασυνδέσεις ανάμεσα σε αυτές τις οντότητες (και τις επιμέρους κατηγοριοποιήσεις τους ανάλογα με τον επιστημονικό φακό που τις παρατηρεί) είναι πολυάριθμες και εξαιρετικά πολύπλοκες. Προσωπικοί υπολογιστές, ενοποιημένα πληροφοριακά συστήματα διαφόρων μεγεθών και κρισιμότητας, και κυρίως οι χρήστες τους ενώνονται σε μια ροή πληροφοριών μεταξύ συστημάτων με τη χρήση πολλών τρόπων και μεθόδων, ενώ πρακτικά καθημερινά εμφανίζονται και νέοι ανταγωνιστικοί τρόποι πρόσβασης προς τη ζητούμενη πληροφορία. Όλα αυτά συμβαίνουν σε ένα κοινό «πεδίο μάχης» τον Κυβερνοχώρο, βασικό «έδαφος» του οποίου αποτελεί το Διαδίκτυο. Με δεδομένο πως στα συστήματα ρέει [1] ενέργεια, ύλη και πληροφορία ανάμεσα στα υποσυστήματα που τα αποτελούν, αλλά και ανάμεσα στα ίδια τα συστήματα που επικοινωνούν μέσω των διασυνδέσεών τους στον Κυβερνοχώρο και το Διαδίκτυο, η ανάγκη για τον έλεγχο της ροής της διακινούμενης πληροφορίας αυξάνεται.

Οι προσπάθειες ελέγχου της ροής της διακινούμενης πληροφορίας υλοποιούνται με την υιοθέτηση και εφαρμογή πολιτικών ασφαλείας. Η διαρκώς αυξανόμενη πολυπλοκότητα των συστημάτων που είναι διαθέσιμα στον χρήστη ώστε να μπορέσει να επιτελέσει το έργο που επιθυμεί, η αυτοματοποίηση και συστηματοποίηση διαδικασιών που παλαιότερα διεκπεραιώνονταν χειρονακτικά, η διασύνδεση κρίσιμων υποδομών σε αυτό το δίκτυο συνεργασίας και λειτουργίας, οι νέες ανάγκες που αναδραστικά δημιουργούνται σε αυτό το υπέρ-δικτυωμένο περιβάλλον, έχουν σαν αποτέλεσμα την δημιουργία πολιτικών ασφαλείας που είναι συνήθως πολύπλοκες και που η εμπειρία δείχνει πως είναι δύσκολο πολλές φορές να γίνουν κατανοητές από χρήστες και διαχειριστές, καθιστώντας ουσιαστικά την εφαρμογή τους αν όχι ανέφικτη, τουλάχιστον προβληματική.

Σε αυτό το δυστοπικό περιβάλλον πρέπει να συμπεριληφθεί υπόψη πως μεσαίου και μεγάλου μεγέθους οργανισμοί ήδη έχουν κατανεμημένα δίκτυα στα οποία διαφορετικές νησίδες

οφείλουν να υπακούουν σε διαφορετικούς κανόνες ροής της πληροφορίας, όχι μόνο σε ότι αφορά στα συστήματα του οργανισμού με «τον έξω κόσμο», αλλά και εσωτερικά μεταξύ τους καθώς οι ίδιοι οι οργανισμοί διαβαθμίζουν την διακινούμενη πληροφορία και στο εσωτερικό τους. Πλήθος βιβλιογραφίας (ενδεικτικά [2] και [3]) ασχολείται με τον ορισμό των κατάλληλων εκείνων οντοτήτων που περιγράφουν έναν οργανισμό και τα προστατευόμενα στοιχεία του, τους ρόλους πρόσβασης στην πληροφορία ανάλογα με το είδος της (αδιαβάθμητη, διάφορα είδη διαβαθμίσεων κ.ο.κ.), την απόδοση ρόλων σε φυσικά πρόσωπα και τις πολιτικές ασφαλείας που αφορούν την λειτουργία των συστημάτων ενός οργανισμού.

Η ορθή τόσο τεχνικά, όσο και νομικά, διαχείριση τέτοιων κατανεμημένων συστημάτων γίνεται ακόμα πιο πολύπλοκη από το γεγονός πως πρόκειται για συστήματα ετερογενή, τόσο σε αρχιτεκτονική, όσο και σε πλατφόρμες υλοποίησης, συστήματα τα οποία εξελίσσονται στο χρόνο, δεν έχουν πάντα κεντρικό σχεδιασμό ή κατασκευαστή, κληρονομούνται στο πέρασμα του χρόνου ως ad-hoc συστήματα που κάνουν καλά τη δουλειά τους και κανείς δεν τολμά να τα αλλάξει, βελτιώσει, προσαρμόσει σε πιο σύγχρονα μοντέλα, ενώ ακόμα και η ίδια η πολιτική κατάσταση εντός ενός οργανισμού ή και οι διαπροσωπικές σχέσεις μπορεί να επηρεάζουν την πρόσβαση, συντήρηση, εξέλιξη και διαχείρισή τους. Για τον διαχειριστή ασφαλείας τέτοια ζητήματα αυξάνουν την πολυπλοκότητα του δημιουργήματος που πρέπει να χειριστεί σημαντικά, ειδικά όταν πρέπει να διατηρήσει τα συστήματα ασφαλή κρατώντας τα πάντα ευθυγραμμισμένα με τους επιχειρησιακούς στόχους του οργανισμού.

Η παρακολούθηση λοιπόν των ζητημάτων που προκύπτουν σε ένα περιβάλλον που εργάζεται 24x7 γίνεται ένα σισύφειο έργο καθώς συνήθως ο χρόνος δεν είναι ποτέ αρκετός για το συνήθως ελλιπές σε αριθμό (και προϋπολογισμό) προσωπικό επιφορτισμένο με την ασφάλεια των συστημάτων. Καθώς για τις διοικήσεις των οργανισμών πρόκειται για cost centers, μια και δεν συνεισφέρουν άμεσα στη εισροή κεφαλαίων, είναι πάντοτε δύσκολο να βελτιωθούν οι συνθήκες παρακολούθησης, παρόλο που αρκετές φορές η διοίκηση καταλαβαίνει το πρόβλημα. Το γεγονός αυτό έχει σαν αποτέλεσμα την ανάπτυξη εσωτερικών συστημάτων παρακολούθησης τα οποία δεν είναι πάντα σε θέση να ακολουθήσουν την πορεία του οργανισμού.

Η αγορά επίσης έχει προσπαθήσει να απαντήσει σε αυτό το πρόβλημα, ειδικά στις περιπτώσεις της εφαρμογής πολιτικών ασφαλείας στο κατώτερο επίπεδο, δηλαδή στον τελικό εξοπλισμό. Αυτό έχει γίνει με τρόπο που συμφέρει τους παίκτες της αγοράς, ειδικά τους κατασκευαστές εξοπλισμού οι οποίοι προσφέρουν ενοποιημένες λύσεις για τον δικό τους εξοπλισμό με περιορισμένη έως καθόλου δυνατότητα συνεργασίας με εξοπλισμό ανταγωνιστών. Στα ετερογενή περιβάλλοντα που αποτελούν τόσο τους οργανισμούς όσο και τα οικιακά ακόμα δίκτυα, π.χ. PC, iPad, λοιπές δικτυωμένες συσκευές τέτοιες λύσεις είναι περιορισμένης εμβελείας. Η αγορά έχει επίσης προσπαθήσει να απαντήσει σε αυτό το πρόβλημα με προσφερόμενες λύσεις που μπορούν να διαχειριστούν εξοπλισμό από περισσότερους από έναν κατασκευαστές. Η πληθώρα όμως των κατασκευαστών και των εγκαταστάσεων εξοπλισμού τους, πολλές από τις οποίες μπορεί μεν να χαρακτηρίζονται σαν end-of-life (δηλαδή χωρίς καμία υποστήριξη πλέον) από τον κατασκευαστή, αλλά είναι σε πλήρη παραγωγική λειτουργία από τον ιδιοκτήτη, όπως και οι πολλές διαφορετικές εκδόσεις

λογισμικού πάνω στον εξοπλισμό αυτό, κάνουν ακόμα και αυτές τις λύσεις προβληματικές (ακόμα και όταν το οικονομικό τους βάρος δεν είναι πρόβλημα).

Πρέπει να διευκολυνθεί λοιπόν η διαχείριση των πολιτικών λειτουργίας και ασφαλείας ενός οργανισμού, τόσο αυτών που είναι καταγεγραμμένες, όσο και αυτών που ενώ είναι άγραφες είναι αυτές που τηρούνται, σε ένα τέτοιο ολοένα και περισσότερο πολύπλοκο περιβάλλον, με συνεχώς περισσότερη διακινούμενη πληροφορία η οποία χαρακτηρίζεται σημαντική. Για να γίνει κάτι τέτοιο όχι μόνο εφικτό αλλά και αποδείξιμο πέρα από τη χρήση επιχειρημάτων της μορφής "best current practice" είναι απαραίτητη η χρήση μαθηματικών εργαλείων που θα μπορούν να προσφέρουν μετρήσιμα αυτή την εξασφάλιση σε όλους τους εμπλεκόμενους στη ροή της πληροφορίας, χρήστες, διαχειριστές συστημάτων, υπεύθυνους ασφαλείας, στελέχη και διοίκηση.

Ένα τέτοιο εργαλείο που μπορεί να χρησιμοποιηθεί για την ανάπτυξη συστημάτων είναι ο Λογισμός Γεγονότων (Event Calculus) [7] ο οποίος ιστορικά προέρχεται από το χώρο της Τεχνητής Νοημοσύνης και της Ρομποτικής. Στην διατριβή αυτή περιγράφεται η πλέον πετυχημένη χρήστη του Λογισμού Γεγονότων [12] στο χώρο των πολιτικών ασφαλείας, επισημαίνονται βασικά προβλήματα που δεν επιτρέπουν ακόμη την ευρύτερη χρήση του, αναλύεται η λειτουργία της βασικής υλοποίησης Λογισμού Γεγονότων σε PROLOG και προτείνεται μια μέθοδος απλοποίησης που μπορεί να ωθήσει προς την υιοθέτησή της από περισσότερα συστήματα ανάπτυξης και ελέγχου πολιτικών ασφαλείας στο μέλλον.

Η ανάπτυξη ενός πληροφοριακού συστήματος έχει ως σκοπό την δημιουργία ανταγωνιστικού πλεονεκτήματος στον ιδιοκτήτη. Για να συμβεί αυτό στο πληροφοριακό σύστημα αποτυπώνονται οι λογικοί επιχειρησιακοί κανόνες που εφαρμόζει ο ιδιοκτήτης για τη λειτουργία του. Προφανώς κατά την ανάπτυξη του συστήματος, τόσο το σύστημα, όσο και οι κανόνες λειτουργίας που αποτυπώνει προσαρμόζονται στη νέα κοινή πραγματικότητα συμβίωσης που διαμορφώνεται στον οργανισμό. Επομένως στον πυρήνα του ένα πληροφοριακό σύστημα είναι ένα σύστημα χειρισμού κανόνων και λογικής το οποίο συνήθως αναπτύσσεται με εργαλεία τα οποία δεν είναι τέτοια που να βοηθούν στον χειρισμό αυτό, επιβάλλοντας πρόσθετο προγραμματιστικό κόπο (και άρα πρόσθετο ρίσκο για λάθη ανάπτυξης και υλοποίησης). Το φαινόμενο αυτό έχει παρατηρηθεί από πολλούς μελετητές και έχει άριστα αποδοθεί από τον Greenspun [4] ο οποίος στον διάσημο «$10^{ο}$ κανόνα» του, ορίζει πως:

> *«Κάθε ικανά περίπλοκο πρόγραμμα γραμμένο σε C ή σε FORTRAN περιέχει μια ad-hoc, ατεκμηρίωτη, γεμάτη λάθη, αργή υλοποίηση της Common Lisp»*

Καθώς όπως έχε ήδη γραφεί, ο πυρήνας των πληροφοριακών συστημάτων είναι ο χειρισμός λογικής (επιχειρηματικοί κανόνες) ο κανόνας ακολουθείται από το λεγόμενο PROLOG follow-up:

> *«Κάθε ικανά περίπλοκο πρόγραμμα γραμμένο σε Lisp θα περιέχει μια αργή υλοποίηση της PROLOG»*

Η ανάπτυξη ενός πληροφοριακού συστήματος έχει ως στόχο να αποτυπώσει και να διευκολύνει με τη λειτουργία του τους κανόνες που διέπουν τον ιδιοκτήτη οργανισμό. Στο πολύπλοκο σημερινό περιβάλλον με κατανεμημένα συστήματα, κανόνες (και εξαιρέσεις τους, οι οποίες μπορεί να έχουν και περιορισμένη χρονική διάρκεια) ένα πληροφοριακό σύστημα διαχειρίζεται την είσοδο που δέχεται από τους χρήστες του (οι οποίοι στην πραγματικότητα αποτελούν και αυτοί μέρος του συστήματος), την έξοδο που αποδίδει, τα δεδομένα τα οποία επεξεργάζεται και τους κανόνες που διέπουν όλα τα παραπάνω.

Τόσο η Lisp, όσο και η PROLOG, αποτελούν κατεξοχήν γλώσσες προγραμματισμού οι οποίες έχουν δημιουργηθεί για το χειρισμό κανόνων και λογικής και άρα είναι ευνόητο πως θα μπορούσαν να χρησιμοποιηθούν και για την επεξεργασία και διαχείριση κανόνων επιχειρησιακής λογικής. Το ψυχολογικό όμως φράγμα μάθησης που έχει δημιουργηθεί γύρω από αυτά τα εργαλεία προγραμματισμού για δεκαετίες τα κάνει να αποτελούν τις τελευταίες επιλογές για αυτούς που αναπτύσσουν πληροφοριακά συστήματα, ακόμα και για εκείνα τα υποσυστήματά τους στα οποία πραγματοποιείται η επαλήθευση των επιχειρησιακών κανόνων.

Η παρατήρηση του Greenspun λοιπόν (ο οποίος έχει εκφράσει μόνο αυτόν τον κανόνα, η χρήση του αριθμού 10 έχει επιλεγεί για έμφαση στην παρατήρηση) έχει να κάνει με το γεγονός πως κάθε πληροφοριακό σύστημα εξαιτίας της πολυπλοκότητάς του, τελικά στον πυρήνα του υλοποιεί μια μηχανή επαλήθευσης κανόνων που ομοιάζει είτε με τη Lisp, είτε με την PROLOG, ακόμα και εάν αυτό δεν το έχουν κατά νου οι δημιουργοί του.

Από το γεγονός αυτό δεν ξεφεύγουν ούτε οι προσφερόμενες λύσεις στην αγορά για την κατασκευή, υλοποίηση και διαχείριση πολιτικών ασφαλείας. Η ανάπτυξη του απαραίτητου λογισμικού γίνεται χρησιμοποιώντας εργαλεία που είναι καταρχήν προσιτά για την κατασκευή διεπαφής χρήστη και δευτερευόντως για τη διαχείριση λογικής όπως συμβαίνει με τις γλώσσες προγραμματισμού που δεν ακολουθούν το συναρτησιακό ή το λογικό μοντέλο. Σε αυτό προστίθεται και το μαθησιακό κενό σε σχέση με τα συγκεκριμένα μοντέλα, όπως και η έλλειψη μαθηματικής σκέψης σε αρκετούς προγραμματιστές [5].

Περιγράφοντας επομένως μια μεθοδολογία χρήσης της PROLOG [6], σε συνδυασμό με τον μαθηματικό φορμαλισμό του Λογισμού Γεγονότων μπορεί να διευκολυνθεί η δημιουργία εργαλείων που θα χρησιμοποιούν αυτόν τον ισχυρό μαθηματικά πυρήνα, για την περιγραφή, έλεγχο αλλά και υλοποίηση ακόμα και στο κατώτερο επίπεδο των πολιτικών ασφαλείας σε ένα διαρκώς μεταβαλλόμενο κατανεμημένο περιβάλλον, όπως είναι αυτό που αντιμετωπίζουν καθημερινά οι διαχειριστές συστημάτων και οι διαχειριστές ασφαλείας, ερχόμενοι συχνά αντιμέτωποι με τις πιο αντιφατικές απαιτήσεις.

Αλλαγές και Χρόνος

Οι πολιτικές ασφαλείας εντάσσονται σε ένα γενικότερο πλαίσιο κανόνων που ελέγχουν τη ροή της πληροφορίας σε ένα δυναμικό σύστημα το οποίο μεταβάλλεται με το χρόνο. Η μεταβολή και προσαρμογή ενός συστήματος σε νέα περιβάλλοντα και καταστάσεις είναι απαραίτητη για την συνέχιση της λειτουργίας του, διαφορετικά είτε το σύστημα θα αντικατασταθεί από ένα νέο που θα μπορεί να προσαρμοστεί στις νέες καταστάσεις, είτε ο οργανισμός δεν θα μπορεί να ανταπεξέλθει στο νέο περιβάλλον και θα διαταραχθεί σημαντικά η λειτουργία του, αν όχι και η ίδια η ύπαρξή του. Αυτό συμβαίνει και με τις πολιτικές ασφαλείας. Οφείλουν να μεταβάλλονται διαρκώς και να προσαρμόζονται στις νέες συνθήκες που προκύπτουν (και που δεν είναι εύκολα προβλέψιμες), να εξυπηρετούν τη λειτουργία του συστήματος, να προστατεύουν τα κρίσιμα σημεία και να μην δυσκολεύουν τους χρήστες. Μια πολιτική ασφαλείας δεν μπορεί να είναι το απαράβατο ευαγγέλιο που ορίζει τι επιτρέπεται και τι όχι σε βάρος της επιχειρηματικής λειτουργίας του οργανισμού.

Η αναζήτηση λοιπόν εργαλείων χειρισμού λογικών κανόνων πρόσβασης, είναι απαραίτητο να περιλαμβάνει μαθηματικά μοντέλα τα οποία να μπορούν να διαχειρίζονται το χρόνο. Δύο τέτοια μοντέλα είναι η Situation Calculus (η οποία περιγράφεται στο πρώτο μισό του [8]) που δημιουργήθηκε από τον πατέρα της Lisp, John McCarthy, και η Event Calculus ([7], [8] και [9]) που δημιουργήθηκε από τους Kowalski και Sergot. Η Situation Calculus διαχειρίζεται καθολικές καταστάσεις σε ένα σύστημα, ενώ η Event Calculus διαχειρίζεται τοπικά γεγονότα και χρονικά διαστήματα. Και οι δύο εντάσσονται στην προσπάθεια επίλυσης ενός προβλήματος που επί μακρύ χρονικό διάστημα απασχόλησε την κοινότητα της Τεχνητής Νοημοσύνης, το Πρόβλημα Πλαισίου (Frame Problem).

ΠΡΟΒΛΗΜΑ ΠΛΑΙΣΙΟΥ

Η κατανόηση του Προβλήματος Πλαισίου είναι απαραίτητη όχι τόσο γιατί αποτελεί αντικείμενο του πεδίου της ασφάλειας των υπολογιστικών συστημάτων, όσο γιατί ο χώρος αυτός δανείζεται

ένα εργαλείο που το επιλύει για να δώσει απαντήσεις σε δικά του προβλήματα. Η επίλυση του Προβλήματος Πλαισίου εντάσσεται στο γενικότερο πλαίσιο της Συλλογιστικής Κοινής Λογικής.

Η Συλλογιστική Κοινής Λογικής (Commonsense Reasoning) είναι μια διαδικασία μέσω της οποίας χρησιμοποιείται πληροφορία για μια κατάσταση στον «κόσμο» (αρχικό σενάριο) ώστε να καταλήξει σε συμπεράσματα για μια τελική κατάσταση, με βάση το σώμα της «κοινής λογικής» (όχι όπως αυτή χρησιμοποιείται στην καθομιλουμένη, αλλά όπως χρησιμοποιείται στο χώρο της Τεχνητής Νοημοσύνης για να δηλώσει συμπεράσματα στα οποία ο άνθρωπος φτάνει «αυτόματα» ενώ ένα πρόγραμμα χρειάζεται κοπιώδη δουλειά περιγραφής του κόσμου ώστε οριακά να τα πλησιάσει).

Παρόλο που ο άνθρωπος ενστικτωδώς χρησιμοποιεί το Commonsense Reasoning, ως σύνολο ενεργειών κάθε άλλο παρά απλό είναι. Αυτό γίνεται κατανοητό όταν προσπαθεί κανείς να περιγράψει υπολογιστικά μια σειρά από γεγονότα και συμπεράσματα όπως αυτά που παρουσιάζονται στα παραδείγματα του [9]:

- Ενώ ήταν στο σαλόνι, η Λίζα πήρε την εφημερίδα και πήγε στην κουζίνα. Πού κατέληξε η εφημερίδα; Στην κουζίνα.
- Η Κατερίνα άφησε ένα βιβλίο στο τραπεζάκι του σαλονιού και βγήκε από το δωμάτιο. Όταν επέστρεψε το βιβλίο δεν ήταν εκεί. Τι έγινε το βιβλίο; Κάποιος πρέπει να το πήρε.
- Ο Γιάννης κλείνει το νεροχύτη, ανοίγει τη βρύση και βγαίνει από το δωμάτιο. Τι θα συμβεί; Θα γεμίσει ο νεροχύτης και μετά το νερό θα πλημυρίσει το δωμάτιο.
- Η Άννα ανοίγει τον ανεμιστήρα. Τι θα συμβεί; Εάν ο ανεμιστήρας είναι στην πρίζα θα δουλέψει, διαφορετικά δεν θα γίνει τίποτε.
- Μια γάτα είδε φαγητό στο τραπέζι. Η γάτα πηδάει σε μια κοντινή καρέκλα. Τι θα κάνει η γάτα; Θα πηδήξει από την καρέκλα στο τραπέζι για να φάει.

Για τη δημιουργία μιας αυτόματης συλλογιστικής Commonsense Reasoning πρέπει να αντιμετωπιστούν τα ακόλουθα προβλήματα:

- Αναπαράσταση: Η μέθοδος πρέπει να είναι σε θέση να αναπαριστά σενάρια του κόσμου και της γνώσης που υπάρχει για τα σενάρια αυτά.
- Οντότητες: Η μέθοδος πρέπει να αναπαριστά οντότητες, γεγονότα, χρόνο και ιδιότητες που μεταβάλλονται με το χρόνο.
- Πεδία εφαρμογής: Η μέθοδος πρέπει να αναπαριστά και να εξάγει συμπεράσματα για το χρόνο, το χώρο και τις καταστάσεις. Πρέπει επίσης να μπορεί να διαχωρίζει τα αντικείμενα με βάση την ταυτότητά τους.
- Αντιμετώπιση φαινομένων: Η μέθοδος πρέπει να μπορεί να χειρίζεται την Αρχή της Αδράνειας (τα αντικείμενα μένουν ανεπηρέαστα από γεγονότα εκτός κι αν τα γεγονότα ειδικά τα επηρεάζουν). Πρέπει να μπορεί να χειρίζεται ταυτόχρονα γεγονότα με αθροιστικά ή ακυρωτικά αποτελέσματα (ένα γεγονός να ακυρώνει ένα άλλο, π.χ. ο οδηγός αφήνει το φρένο αλλά έχει βάλει χειρόφρενο), να μπορεί να

χειρίζεται (καταλαβαίνει) αρχικές συνθήκες ή και γεγονότα που εκκινούνται λόγω άλλων συμβάντων.

- Εξαγωγή συμπερασμάτων: Η μέθοδος πρέπει να ορίζει διαδικασίες για την εξαγωγή συμπερασμάτων με τη χρήση όλων των παραπάνω.

Τι είναι fluent:

Στην Τεχνητή Νοημοσύνη ως fluent ορίζεται ([9], [11]) μια ιδιότητα που μεταβάλλεται με το χρόνο, όπως π.χ. η θέση ενός φυσικού αντικειμένου. Ο όρος fluent χρησιμοποιήθηκε από τον McCarthy για να ορίσει το κατηγόρημα ή τη συνάρτηση του οποίου το όρισμα είναι μια κατάσταση. Ο McCarthy δανείστηκε με τη σειρά του τον όρο από τον Νεύτωνα ο οποίος τον χρησιμοποιούσε για να χαρακτηρίσει ποσότητες που άλλαζαν με τον χρόνο.

Ορισμός του Προβλήματος Πλαισίου:

Σύμφωνα με τα [8] και [9] το πρόβλημα της αναπαράστασης και της εξαγωγής συμπερασμάτων για τα οποία ένα fluent δεν αλλάζει όταν συμβαίνει ένα γεγονός, ονομάζεται Πρόβλημα Πλαισίου. Η ονομασία αυτή δόθηκε στο πρόβλημα από τον McCarthy επειδή την περίοδο που διαμόρφωνε την προβληματική αυτή μελετούσε ένα βιβλίο γεωμετρίας.

Υπό αυτές τις συνθήκες το Πρόβλημα Πλαισίου είναι το πρόβλημα που παρουσιάζεται όταν γίνεται προσπάθεια να αναπαρασταθούν αποτελέσματα ενεργειών και γεγονότων με τη χρήση Τυπικής Λογικής [8]. Καθώς έχει ήδη αναφερθεί πως στόχος αυτής της εργασίας είναι η χρήση εργαλείων Μαθηματικής Λογικής για την επεξεργασία πολιτικών ασφαλείας (δηλαδή ενός συνόλου ενεργειών και κανόνων που αφορούν γεγονότα και αποτελέσματα ενεργειών) η σύνδεση με το Πρόβλημα Πλαισίου γίνεται αντιληπτή καθώς σε ένα υπολογιστικό σύστημα η εφαρμοζόμενη πολιτική ασφαλείας οφείλει να διαχειριστεί ιδιότητες μεταβαλλόμενες με το χρόνο, γεγονότα που προκαλούν ή όχι αποτελέσματα και επόμενες ενέργειες.

Λογισμός Γεγονότων

Ο Λογισμός Γεγονότων είναι μια τυπική γλώσσα που μπορεί να χρησιμοποιηθεί για την εξαγωγή συμπερασμάτων για δυναμικά συστήματα, όπως αυτά που αποτελούν το αντικείμενο εφαρμογής των πολιτικών ασφαλείας. Χειρίζεται fluent, γεγονότα (από όπου προέρχεται και η ονομασία) και σημεία στο χρόνο. Τα κατηγορήματα και αξιώματα του Λογισμού Γεγονότων όπως παρουσιάζονται στο [12] είναι:

Event Calculus

Βασικά Κατηγορήματα:	
$Initiates(A,B,T)$	Το γεγονός A εκκινεί το fluent B για χρόνο t > T
$Terminates(A,B,T)$	Το γεγονός A τερματίζει το fluent B για χρόνο t > T
$Releases\ A,B,T$	Για το fluent B δεν ισχύει ο νόμος της αδράνειας μετά το γεγονός A τη χρονική στιγμή T
$Happens(A,T_1,T_2)$	Το γεγονός A συμβαίνει τη χρονική στιγμή T_1 και T_2
$Happens(A,T) \equiv Happens(A,T,T)$	
$HoldsAt(B,T)$	Το fluent B ισχύει τη χρονική στιγμή T
$Initially_P(B)$	Το fluent B είναι αρχικά αληθές
$Initially_N(B)$	Το fluent B είναι αρχικά ψευδές
Βοηθητικά Κατηγορήματα	
$Clipped(T_1,B,T_2)$	Το fluent B τερματίζει κάποια στιγμή ανάμεσα στα T_1 και T_2
$Declipped(T_1,B,T_2)$	Το fluent B αρχικοποιείται κάποια στιγμή ανάμεσα στα T_1 και T_2
Ανεξάρτητα αξιώματα	
$HoldsAt\ B,T_1 \leftarrow HoldsAt\ B,T \wedge !Clipped(T,B,T_1) \wedge T < T_1$	
$HoldsAt\ B,T_1 \leftarrow Initiates\ A,B,T \wedge Happens\ A,T \wedge !Clipped(T,B,T_1) \wedge T < T_1$	
$!HoldsAt\ B,T_1 \leftarrow !HoldsAt\ B,T \wedge !Declipped(T,B,T_1) \wedge T < T_1$	
$!HoldsAt\ B,T_1 \leftarrow Terminates\ A,B,T \wedge Happens\ A,T \wedge !Declipped(T,B,T_1) \wedge T < T_1$	
Σημείωση που αφορά τον παραπάνω πίνακα: Ο λογικός τελεστής NOT συμβολίζεται με το θαυμαστικό (!) και όχι με το σύμβολο ¬ καθώς δεν περιέχεται στον Equation Editor του Word 2007.	

Ο παραπάνω φορμαλισμός είναι ο κλασικός φορμαλισμός του Λογισμού Γεγονότων στον οποίο οι θεωρίες αποτυπώνονται ως προτάσεις Horn. Ο McCarthy έλυσε το Πρόβλημα Πλαισίου με τη χρήση του μηχανισμού του circumscription ([13] και [14]). Με τη χρήση αυτής της μεθόδου είναι δυνατή η πλήρωση των κατηγορημάτων *Initiates*, *Terminates* και *Happens*, ενώ ταυτόχρονα τα κατηγορήματα *HoldsAt*, *Initially*$_P$ και *Initially*$_N$ παραμένουν ανοιχτά. Με την προσέγγιση αυτή είναι δυνατή η χρήση μερικού γνωστικού αντικειμένου (partial domain knowledge) για την αναπαράσταση π.χ. των αρχικών συνθηκών του προβλήματος. Σε αυτό το σημείο πρέπει να σημειωθεί πως ο φορμαλισμός του παραπάνω πίνακα δεν είναι ο μοναδικός τρόπος έκφρασης του Λογισμού Γεγονότων. Το [9] περιγράφει λεπτομερώς εναλλακτικές αξιωματικές θεμελιώσεις. Για τον αναγνώστη που ενδιαφέρεται για μία σύντομη παρουσίαση των θεμελιώσεων αυτών υπάρχει το [15] το οποίο βασίζεται στα [9], [16], [17] και [18].

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Υπόθεση Κλειστού Κόσμου (Closed World Assumption)

Στο έμπειρο μάτι του αναγνώστη είναι φανερή η σύνδεση της PROLOG με τον Λογισμό Γεγονότων αν και δεν είναι ευθεία ή τόσο άμεση όσο θα επιθυμούσε ο ανυπόμονος προγραμματιστής χωρίς το απαραίτητο υπόβαθρο στη Λογική. Απαραίτητη προϋπόθεση για να είναι δυνατό να επιτευχθεί Συλλογιστική Κοινής Λογικής με τη χρήση της PROLOG [22] είναι η κατανόηση της Υπόθεσης Κλειστού Κόσμου η οποία μελετήθηκε αρχικά στο [21].

Με βάση την υπόθεση αυτή το σύνολο των διαθέσιμων αξιωμάτων περιέχει όλη την απαραίτητη πληροφορία. Για τις ανάγκες του negation-as-failure [24] χρησιμοποιείται η υπόθεση του κλειστού κόσμου για οποιοδήποτε κατηγόρημα δεν μπορεί να αποδειχτεί. Εάν μια ατομική φόρμουλα δεν μπορεί να αποδειχθεί με τα διαθέσιμα γνωστά αξιώματα, τότε η άρνησή της θεωρείται αληθής. Επομένως για να μπορεί κάποιος να εξετάσει την «βάση» των αξιωμάτων χρειάζεται μόνο «θετική» πληροφόρηση σε αυτή. Η πληροφόρηση που αφορά τις αρνήσεις εξάγεται αυτόματα χάρη στο negation-as-failure. Αυτό το χαρακτηριστικό είναι σημαντικό καθώς μπορεί να βελτιώσει σημαντικά την πολυπλοκότητα τόσο του συμπερασματικού μηχανισμού όσο και της αναπαράστασης της απαιτούμενης γνώσης. Η PROLOG σαν υποσύνολο της λογικής πρώτης τάξης με τη μορφή προτάσεων Horn δεν μπορεί αυτόματα να εκφράσει αρνητικά συμπεράσματα. Όμως με τη χρήση του «κλειστού κόσμου» αυτό αλλάζει και μάλιστα τα μοντέλα που προκύπτουν είναι Herbrand [23].

Η αρνητική πληροφορία (negative information) εξακολουθεί να είναι απαραίτητη για την διατήρηση της ακεραιότητας της πληροφορίας που βρίσκεται στο σύστημα. Αυτό σημαίνει πως για κάθε κατηγόρημα που περιέχεται στο σύστημα θα πρέπει να υπάρχει και το αντίστοιχο κατηγόρημα που θα περιέχει την άρνησή του. Είναι όμως δυνατό να αποφευχθεί αυτό με την χρήση ενός κατηγορήματος που θα έχει ως στόχο την ακεραιότητα της πληροφορίας ως εξής:

```
support(a,b).
consistent(support(A,B)) :- not(support(a,b)).
```

Επομένως εάν το σώμα μιας γραμμής PROLOG περιέχει την προτάσεις της μορφής *not(ab)* τότε έχει επιτευχθεί ένας σημαντικός στόχος: Τα αντικείμενα θεωρούνται κανονικά εφόσον δεν υπάρχουν στοιχεία για το αντίθετο. Αυτό έρχεται σε συμφωνία και με την Αρχή της Αδράνειας που ισχύει στα συστήματα που εξετάζουμε και που ορίζει πως:

Ένα γεγονός δεν επηρεάζει κάποιο αντικείμενο, παρά μόνο όταν ρητά το επηρεάζει.

Το παραπάνω είναι βοηθητικό πάντοτε στην εξαγωγή συμπερασμάτων σχετικών με την ευστάθεια και την ασφάλεια υπολογιστικών αλλά και άλλων συστημάτων.

Ο Λογισμός Γεγονότων υποστηρίζει αφαιρετικό (deductive), επαγωγικό (inductive) και απαγωγικό (abductive) συλλογισμό [25].

- Ο αφαιρετικός συλλογισμός επιτρέπει να συμπεράνουμε το Β από το Α, μόνο όταν το Β είναι μια συνέπεια του Α. Είναι δηλαδή η διαδικασία της εξαγωγής των συμπερασμάτων από όσα έχουν αρχικά υποτεθεί. Με δεδομένη την αλήθεια των υποθέσεων, μια ορθή συλλογιστική εγγυάται την αλήθεια του αποτελέσματος. Για παράδειγμα, με βάση την υπόθεση πως όλοι οι εργένηδες είναι ανύπανδροι άνδρες, τότε εάν κάποιος είναι εργένης, είναι ανύπανδρος άνδρας.

- Ο επαγωγικός συλλογισμός επιτρέπει να συμπεραίνουμε το Β από το Α όταν το Β δεν προκύπτει ευθέως ως συνέπεια του Α. Το Α δίνει ισχυρούς λόγους για να ισχύει το Β, αλλά δεν αρκεί. Για παράδειγμα, εάν όλοι οι κύκνοι που έχει παρατηρήσει κάποιος είναι λευκοί, τότε και ο επόμενος κύκνος που θα παρατηρηθεί αναμένεται να είναι λευκός. Το αποτέλεσμα όμως αυτής της πρόγνωσης δεν μπορεί να είναι εγγυημένο (υπάρχουν και μαύροι κύκνοι).

- Ο απαγωγικός συλλογισμός μας επιτρέπει να συμπεραίνουμε το Α ως μια εξήγηση του φαινομένου που οδήγησε στο Β. Αυτή είναι και η διαφορά της απαγωγής από τον αφαιρετικό συλλογισμό, καθώς πρέπει να προσδιοριστεί ο κανόνας από τον οποίο προκύπτει το Β. Αυτό έχει σαν αποτέλεσμα η απαγωγή να είναι το φορμαλιστικό ισοδύναμο της λογικής πλάνης καθώς μία τελική συνθήκη μπορεί να προκύψει από πολλά διαφορετικά σενάρια αρχικών συνθηκών. Η διαδικασία της απαγωγής είναι σημαντικό κομμάτι της Επιστημονικής Μεθόδου. Υπάρχουν άπειρες δυνατές εξηγήσεις για τις φυσικές διαδικασίες που παρατηρούνται, όμως επιλέγεται ένα (ή πολύ λίγα αριθμητικά) σενάριο που να οδηγεί στο παρατηρούμενο αποτέλεσμα με την ελπίδα να απαλειφθούν κάποιες από τις πιθανές εξηγήσεις.

Η απαγωγή έχει μεγάλη σημασία στην ασφάλεια των υπολογιστικών συστημάτων, Με δεδομένη την περιγραφή της συμπεριφοράς ενός συστήματος, η χρήση της απαγωγής είναι δυνατό να καθορίσει τη σειρά των γεγονότων με βάση την οποία κάποιο ή κάποια fluent θα είναι αληθή. Και τα fluent σε αυτή την περίπτωση είναι οι αρχικές εκείνες συνθήκες που περιγράφουν την ασφάλεια του συστήματός που βρίσκεται υπό εξέταση.

Στη συνέχεια θα εξεταστεί η σχέση αυτή μεταξύ της γλώσσας Ponder [29] η οποία σχεδιάστηκε για την εφαρμογή πολιτικών ασφαλείας και της Event Calculus με τη χρήση απαγωγής [28].

Λογισμός Γεγονότων και Πολιτικές Ασφάλειας

Υπάρχουν πολλές γλώσσες περιγραφής και ελέγχου πολιτικών ασφαλείας οι οποίες είναι προσιτές στο χρήστη είτε μέσω γραφικών περιβαλλόντων (συνήθως τέτοιες είναι οι λύσεις που προκρίνει η αγορά), είτε προγραμματιστικά. Μια γλώσσα καθορισμού πολιτικών που ανήκει στη δεύτερη κατηγορία είναι η Ponder [29] η οποία παρέχει ένα κοινό μέσο για την προδιαγραφή πολιτικών ασφαλείας που με τη σειρά τους αποτυπώνουν πολλούς και διαφορετικούς μηχανισμούς ελέγχου που υλοποιούνται σε ένα πληροφοριακό σύστημα και αφορούν σε λειτουργικά συστήματα, προσωπικούς υπολογιστές, firewalls, βάσεις δεδομένων και λοιπές εφαρμογές Ιστού. Υποστηρίζει πολιτικές οι οποίες ενεργοποιούνται από γεγονότα, κανόνες που ελέγχουν εάν ισχύει μια προϋπόθεση ώστε να συμβεί κάποια απαιτούμενη ενέργεια σε ένα δίκτυο ή σε ένα κατανεμημένο σύστημα. Μπορεί επίσης να χρησιμοποιηθεί για τη διαχείριση ενεργειών που έχουν ως σκοπό την ασφάλεια του συστήματος και ειδικότερα ενέργειες που έχουν να κάνουν με την εγγραφή χρηστών και υπηρεσιών στο σύστημα, αρχείων καταγραφής ενεργειών, έλεγχο (auditing) ειδικά δε ότι αφορά κρίσιμα υποσυστήματα και την πρόσβαση σε αυτά και φυσικά σε παραβιάσεις των αποτυπωμένων πολιτικών ασφαλείας. Ισχυρές έννοιες που περιλαμβάνει η γλώσσα είναι η δυνατότητα απόδοσης ρόλων, οι ομαδικές πολιτικές, η δυνατότητα να περιγραφεί το οργανόγραμμα ενός οργανισμού και έτσι να υπάρξει και η αντίστοιχη ταύτιση των ρόλων απεικονίζοντας τις σχέσεις μεταξύ των συμμετεχόντων στον οργανισμό. Με την δυνατότητα που υπάρχει εγγενώς στη γλώσσα να είναι αυτές οι περιγραφές επαναχρησιμοποιήσιμες, είναι δυνατή η χρήση της σε μεγάλα πληροφοριακά συστήματα. Η Ponder είναι μία γλώσσα δηλωτική, επεκτεινόμενη και προσαρμοζόμενη σε πολλά περιβάλλοντα. Στο πλαίσιο της διατριβής, δεν εξετάζεται η γλώσσα Ponder, αλλά η σχέση της με τον Λογισμό Γεγονότων και πως αυτή η σχέση μπορεί να βοηθήσει τους διαχειριστές συστημάτων με τη χρήση και άλλων εργαλείων.

Με μια ευρεία ματιά οι πολιτικές ασφαλείας μπορούν να χωριστούν σε πολιτικές αδειοδότησης (authorization) και σε πολιτικές διαχείρισης. Η πρώτη κατηγορία έχει σαν αντικείμενο την πρόσβαση στο σύστημα (access control) και τις απαραίτητες προϋποθέσεις, ενώ η δεύτερη κατηγορία εμπεριέχει τις πολιτικές εκείνες που έχουν να κάνουν με τη συμπεριφορά του συστήματος.

Οι πολιτικές αδειοδότησης πρόσβασης ορίζουν εάν ένα υποκείμενο μπορεί να εκτελέσει μία λειτουργία σε ένα στόχο (αντικείμενο). Σε ένα κλειστό σύστημα, με προεπιλογή την καθολική απαγόρευση της εκτέλεσης όλων των ενεργειών από οποιοδήποτε υποκείμενο προς

οποιοδήποτε στόχο, «θετικές» πολιτικές είναι αυτές που περιγράφουν τις επιτρεπτές ενέργειες που μπορούν να επιτευχθούν. Σε ένα ανοιχτό σύστημα, το οποίο έχει σαν προεπιλογή να επιτρέπονται όλες οι ενέργειες από όλους προς όλους, «αρνητικές» πολιτικές περιγράφουν ότι δεν επιτρέπεται να εκτελεστεί. Αυτό έχει εκφραστεί κατάλληλα από στο [31]:

- That which is not expressly permitted is prohibited.
- That which is not expressly prohibited is permitted.

Σε ένα πληροφοριακό σύστημα ζουν πάντα συνδυασμοί αρνητικών και θετικών πολιτικών πρόσβασης, ανεξάρτητα από το είδος της προεπιλογής. Είναι όμως σημαντικό τόσο για την πολυπλοκότητα, όσο και για τη διαχείριση των κανόνων, να λαμβάνεται υπόψη κατά την υλοποίηση του συστήματος και των κανόνων πρόσβασης η κλειστότητά του.

Οι πολιτικές υποχρέωσης ορίζουν διαδικασίες διαχείρισης οι οποίες πρέπει να εκτελεστούν όταν συμβαίνει ένα συγκεκριμένο γεγονός και κάποιες συμπληρωματικές συνθήκες είναι αληθείς. Ορίζονται σε σχέση με το υποκείμενο το οποίο θα πρέπει να προβεί στις απαραίτητες ενέργειες σε ένα στόχο όταν κάποια συγκεκριμένη συνθήκη είναι αληθής. Αυτές οι πολιτικές είναι εξαρτώμενες από τα γεγονότα και συνεπώς η ύπαρξη ενός από αυτά είναι απαραίτητη συνθήκη ώστε να εκτελεστεί η προβλεπόμενη ενέργεια. Επίσης, ορίζουν πως το υποκείμενο της πολιτικής οφείλει να εκτελέσει την ενέργεια και όχι μόνο απλά να αποφασίσει εάν μια ενέργεια μπορεί να εκτελεστεί ή όχι.

Οι πολιτικές αποχής ορίζονται από τους διαχειριστές ώστε κάτω από ορισμένες συνθήκες συγκεκριμένες ενέργειες να μην πρέπει να εκτελεστούν. Ομοιάζουν με τις αρνητικές πολιτικές πρόσβασης, καθώς όπως είναι φανερό χρησιμοποιούνται και οι δύο με σκοπό να μην εκτελεστεί μια ενέργεια, αλλά υπάρχει μία σημαντική διαφορά: Οι αρνητικές πολιτικές πρόσβασης είναι εργαλείο απόφασης του ελεγκτή του στόχου π.χ. ενός firewall μπροστά από ένα εκτυπωτή. Αντίθετα οι πολιτικές αποδοχής αποτελούν εργαλείο απόφασης του ίδιου του στόχου και μπορούν να χρησιμοποιηθούν όταν το ίδιο το αντικείμενο δεν επιθυμεί να προστατευτεί από τον εκτελούντα την ενέργεια.

Οι πολιτικές είναι αναγκαίο να αναφέρονται όχι μόνο σε συγκεκριμένα αντικείμενα που βρίσκονται στην βάση περιγραφής του κόσμου ο οποίος μοντελοποιεί το υπό εξέταση σύστημα, αλλά οφείλουν να εφαρμόζονται και σε σύνολα αντικειμένων (domains). Με μια τέτοια προσέγγιση κατά την οποία ένα αντικείμενο μπορεί να ανήκει σε ένα σύνολο ή όχι κατά δήλωσή του και μόνο (και ποτέ αυτό να μη συνάγεται συμπερασματικά) είναι δυνατό να ορίζονται ομαδικές πολιτικές που να αφορούν τα αντικείμενα που συμμετέχουν σε ένα σύνολο και για όσο είναι μέλη του συνόλου αυτού. Φυσικά και μπορούν να υπάρχουν πολιτικές που να αναφέρονται στο κενό σύνολο, καθώς ένα domain μπορεί να είναι κενό σε ένα συγκεκριμένο στιγμιότυπο του συστήματος.

Από τα παραπάνω είναι ήδη φανερή η σχέση που μπορεί να αναπτυχθεί ανάμεσα σε πολιτικές ασφαλείας, όπως τις αντιμετωπίζουν γλώσσες διαχείρισης πολιτικών ασφαλείας όπως η Ponder και στο Λογισμό Γεγονότων. Επίσης είναι εύκολο να διαπιστωθεί πως τα συγκεκριμένα τέσσερα ήδη πολιτικών (θετική και αρνητική πρόσβαση, υποχρέωση και αποχή) δεν αφορούν μόνο τη συγκεκριμένη γλώσσα που χρησιμοποιείται στην εργασία αυτή ως παράδειγμα προς εξέταση, αλλά γενικότερα κάθε αντίστοιχη τέτοια προσπάθεια. Επίσης μπορεί να παρατηρήσει κανείς πως οι περισσότεροι μηχανισμοί που αναπτύσσονται, επειδή έχουν να διαχειριστούν κατανεμημένα συστήματα, στα οποία όμως υπάρχει κεντρική διοίκηση (και όχι κάποιας μορφής federation) αγνοούν την υλοποίηση και τον έλεγχο πολιτικών υποχρέωσης και αποχής.

Η σύνδεση με την PROLOG, και το Λογισμό Γεγονότων μπορεί να φανεί εάν παρατηρήσει κανείς πως σε μια πολιτική ασφαλείας υπάρχουν λογικοί κανόνες των οποίων τη συμφωνία οφείλει να ελέγξει ένα σύστημα πριν προβεί στις ενέργειες επίτρεψης / απόρριψης σε σχέση με το ζητούμενο από κάποιο χρήστη. Καθώς επίσης η συμπεριφορά του συστήματος εξαρτάται και από το χρόνο (όπως και οι απαντήσεις του εξαρτώνται και από τη χρονική στιγμή όχι μόνο που γίνονται οι ερωτήσεις, αλλά και πού δίνονται οι απαντήσεις) οι προϋποθέσεις που οφείλουν να είναι αληθείς κατά τον έλεγχο του συστήματος ώστε να πάρει αποφάσεις, αποτελούν τα fluent τα οποία μπορεί κάποιος να ελέγξει με τη χρήση αυτού του μαθηματικού φορμαλισμού όπως αυτός χρησιμοποιείται στο [12].

Καθώς η εφαρμογή πολιτικών ασφαλείας οδηγεί σε ενέργειες που αλλάζουν την κατάσταση του συστήματος, είναι απαραίτητο το μαθηματικό / προγραμματιστικό εργαλείο που θα χρησιμοποιηθεί για να μπορεί, πέρα από την περιγραφή των πολιτικών ασφαλείας, να απεικονίσει και το ίδιο το σύστημα. Για το λόγο αυτό το μοντέλο οφείλει να υποστηρίζει:

- Αντικείμενα και την οργάνωσή τους σε σύνολα (domains)
- Διαχειριστικές ενέργειες που μπορούν να εκτελεστούν και το αποτέλεσμά τους
- Κανόνες πολιτικής

Επί πρόσθετα υπάρχει η ανάγκη για τον καθορισμό κανόνων ανεξάρτητων από την οργάνωση σε σύνολα οι οποίοι αφορούν την επιβολή πολιτικών ασφαλείας. Η ανάγκη μετατροπής από υψηλό επίπεδο σε ένα σύστημα λογικής επιβάλλει την υποστήριξη των ακόλουθων:

- Σταθερές: Έστω πως το σύνολο Obj περιγράφει το σύνολο των αντικειμένων στο σύστημα. Κάθε μέλος του Obj είναι σταθερό.
- Μεταβλητές: Έστω το σύνολο V_O το οποίο περιγράφει τα χαρακτηριστικά των αντικειμένων του Obj και V_P το σύνολο που περιγράφει τις μεταβλητές για τις ενέργειες που υποστηρίζονται από τα αντικείμενα.
- Λειτουργίες / Συναρτήσεις (βλέπε επόμενο πίνακα)
- Κατηγορήματα: Τα κατηγορήματα της Event Calculus και ορισμένα βοηθητικά όπως φαίνονται στον επόμενο πίνακα από το [12].

Λειτουργίες	
$state\ \ Obj, V_O, Value$	Απεικονίζει την κατάσταση μιας μεταβλητής ενός αντικειμένου του συστήματος.
$operation\ \ Obj, Action\ V_P$	Χρησιμοποιείται για να καταδείξει τις ενέργειες που ορίζονται σε λειτουργίες μιας πολιτικής
$systemEvent\ \ Event$	Απεικονίζει οποιοδήποτε γεγονός παράγεται από το σύστημα και ενεργοποιεί πολιτικές υποχρέωσης ή αποχής.
$doAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Απεικονίζει το γεγονός της ενέργειας από το Obj_{Subj} προς το Obj_{Targ}
$requestAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Απεικονίζει το γεγονός που συμβαίνει όταν ο Obj_{Subj} ενεργεί προς τον Obj_{Targ}
$rejectAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Γεγονός που συμβαίνει όταν έχει αποφασιστεί να απορριφθεί μια ενέργεια.
$permit\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Η ενέργεια επιτρέπεται
$deny\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Η ενέργεια απορρίπτεται
$oblig\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Υποχρέωση
$refrain\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P$	Αποχή
Κατηγορήματα	
$object\ \ Obj$	Δήλωση του Obj στο σύστημα
$attr\ \ Obj, V_O$	Δήλωση του V_O ως χαρακτηριστικού του Obj
$method\ \ Obj, Action\ V_p$	Απεικονίζει μια ενέργεια που μπορεί να εκτελέσει το Obj στο σύστημα.
$isDomain\ \ Obj$	Χρησιμοποιείται για να δηλώνει τα σύνολα αντικειμένων στο σύστημα. Ισχύει πως: $object(Obj) \leftarrow isDomain(Obj)$
$isMember\ \ Obj, Dom$	Ισχύει εφόσον $Obj\ \in Dom$

$isSubDomain\ Dom_1, Dom_2$ $\leftarrow isDomain\ Dom_1\ \wedge isDomain\ Dom_2$ $\wedge isMember\ Dom_1, Dom_2\ \wedge Dom_1$ $\neq Dom_2 \wedge !\, isSubDomain\ Dom_2, Dom_1$	Ισχύει εφόσον $Dom_1 \subset Dom_2$. Το κατηγόρημα έχει οριστεί με τρόπο που αποφεύγει τις κυκλικές αναφορές.
$isDerivedMember\ Obj, Dom$ $\leftarrow object\ Obj\ \wedge !\, isDomain\ Obj$ $\wedge isMember\ Obj, Dom$ $isDerivedMember\ Obj, Dom$ $\leftarrow object\ Obj\ \wedge !\, isDomain\ Obj$ $\wedge subDomain\ Dom, SubDom$ $\wedge isDerivedMember\ Obj, SubDom$	Χρησιμοποιείται για να καθοριστεί η σχέση του Obj με όλη την ιεραρχία αντικειμένων και συνόλων στο σύστημα.
$isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ V_P\ \ )$ $\leftarrow object(Obj_{subj}) \wedge object(Obj_{Targ})$ $\wedge method(Obj_{Targ}, Acton\ V_p\ )$	

Παρατηρήσεις:

- Στον αντίστοιχο πίνακα στο [12] υπάρχουν τυπογραφικά λάθη
- Σε πολλές από τις δηλώσεις στον παραπάνω πίνακα εμφανίζεται η πλειάδα (tupple) $(Obj_{Subj}, operation(Obj_{Targ}, Action\ V_P\ )$. Το κατηγόρημα isValidSpec είναι αληθές όταν τα μέλη της πλειάδας είναι συνεπή με τις δηλώσεις που υπάρχουν στο σύστημα.

Από την παρουσίαση των παραπάνω στοιχείων του πίνακα, η σύνδεση ενός συστήματος γραμμένου σε PROLOG με μια μεθοδολογία εξέτασης και εφαρμογής πολιτικών ασφαλείας παύει να είναι απλά διαισθητική (επειδή υπάρχουν κανόνες λογικής που πρέπει να ικανοποιούνται) και γίνεται ξεκάθαρη.

Ένα παράδειγμα

Συνεχίζοντας με το παράδειγμα που δίνεται στο [12], έστω ένας οργανισμός που διαθέτει εκτυπωτές διαμοιρασμένους σε όλες του τις εγκαταστάσεις με στόχο να εξυπηρετεί όλα τα γραφεία. Οι εκτυπωτές είναι οργανωμένοι σε σύνολα με βάση τις ιδιότητές τους, όπως το χρώμα (έγχρωμος, ασπρόμαυρος) η ταχύτητα (πολλές / λίγες σελίδες το λεπτό) και η φυσική τους τοποθεσία. Οι εκτυπωτές διαθέτουν αναγνωριστικά (ονόματα) ώστε να είναι διακριτοί (skyblue, violet, cobalt, gray, crimson, damson). Καθώς οι ιδιότητες των εκτυπωτών μπορούν να οργανωθούν σε σύνολα, δίνεται η περιγραφή του εκτυπωτή crimson στο σύστημα:

```
object(printer-crimson).
attr(printer-crimson, status).
method(printer-crimson, printDoc).
method(printer-crimson, switchPaper).
```

```
isDomain(office).
isDomain(bw-printers).
isDomain(lab).

isMember(lab, office).
isMember(highvol-printers, lab).
isMember(bw-printers, lab).

isMember(printer-crimson, lab).
isMember(printer-crimson, highvol-printers).
isMember(printer-crimson, bw-printers).
```

Περιγράφοντας με αυτό τον τρόπο όλο το σύστημα η βάση της PROLOG έχει πλέον στη διάθεσή της την περιγραφή όλου του συστήματος και μπορεί να απαντήσει ερωτήματα της μορφής:

```
?- isDerivedMember(Printer, office).
```

Ένα τέτοιο ερώτημα επιστρέφει στη μεταβλητή Printer όλους τους εκτυπωτές που ανήκουν στο office.

Εφόσον έχει εισαχθεί η περιγραφή των αντικειμένων στο σύστημα είναι πλέον δυνατή η περιγραφή σε αυτό των ενεργειών που υποστηρίζονται από αυτό καθώς και η συμπεριφορά τους. Με τη χρήση της λειτουργίας *method* απεικονίζονται οι λειτουργίες που υποστηρίζονται από το σύστημα. Για κάθε μία από αυτές είναι απαραίτητος ο ορισμός προϋποθέσεων που πρέπει να ισχύουν πριν και μετά την εκτέλεση της λειτουργίας. Η εκτέλεση κάθε τέτοιας λειτουργίας επιφέρει αλλαγές στην κατάσταση του συστήματος ώστε μετά το τέλος της θα υπάρχουν κάποια νέα fluent που θα ισχύουν, ενώ κάποια άλλα θα έχουν πάψει. Αυτό ακριβώς το λόγο εξυπηρετούν και τα θεμελιώδη κατηγορήματα *initiates* και *terminates*:

$$initiates\ \ doAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ Params\ \ , PostTrue, T_m$$
$$\leftarrow isValidSpec\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ Params\ \ \wedge PreCondition$$

$$terminates(doAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ Params\ \ , PostFalse, T_m)$$
$$\leftarrow isValidSpec\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ Params\ \ \wedge PreCondition$$

Με βάση τον πρώτο κανόνα, όταν ένα γεγονός συμβαίνει (*doAction*) τη χρονική στιγμή T_m εφόσον το *PreCondition* είναι αληθές, τότε και το *PostTrue* θα είναι αληθές μετά την εκτέλεση του γεγονότος. Υπό τις ίδιες προϋποθέσεις το *PostFalse* θα πάψει να ισχύει μετά τη χρονική

στιγμή T_m. Η προϋπόθεση *PreCondition* ορίζεται σαν η λογική σύζευξη κατηγορημάτων *holdsAt*. Τα fluent *PostTrue* και *PostFalse* έχουν οριστεί με τη χρήση κατηγορημάτων *state*. Το κατηγόρημα *isValidSpec* χρησιμοποιείται για να εξασφαλιστεί η συνέπεια με τα περιεχόμενα της γνωσιακής βάσης.

Με βάση τα προηγούμενα φαίνεται πως αυτοί οι κανόνες μπορούν να μοντελοποιήσουν τη συμπεριφορά του συστήματος. Με την υπόθεση πως κάθε εκτυπωτής διαθέτει print manager, για κάθε εκτυπωτή υπάρχουν διαθέσιμες λειτουργίες επισκόπησης της ουράς, προσθήκης και αφαίρεσης εργασιών από αυτή. Επίσης είναι διαθέσιμη διαγνωστική πληροφορία (π.χ. paper jam). Αυτή η παραγόμενη πληροφορία μπορεί να χρησιμοποιηθεί είτε για την αυτόματη διόρθωση προβλημάτων, είτε για την ενημέρωση ενός ανθρώπου-διαχειριστή. Μια τέτοια μοντελοποίηση είναι δυνατό να παρασταθεί με τη χρήση Event Calculus:

$initiates(doAction\ \ printer, operation\ \ printer, switchPaper\ \ , state\ \ printer, status, busy\ \ , T)$ $\leftarrow holdsAt(pos\ \ state\ \ printer, status, busy\ \ , T)$ $\wedge\, holdsAt(pos\ \ statr\ \ printMgr, status, jobSpooled\ \ , T)$
$initiates\ \ doAction\ \ printer, operation\ \ printer, switchPaper\ \ , state\ \ printMgr, status, jobSpooled\ \ , T)$ $\leftarrow holdsAt(pos\ \ state\ \ printer, status, busy\ \ , T)$ $\wedge\, holdsAt(pos\ \ state\ \ printMgr, status, jobspooled\ \ , T)$

Η ανάλυση των πολιτικών που πρέπει να εφαρμοστούν απαιτεί τον προσδιορισμό των αποτελεσμάτων που κάθε πολιτική επιφέρει στην συμπεριφορά του συστήματος. Επομένως επιπλέον της μοντελοποίησης των κανόνων ορισμού των πολιτικών, είναι αναγκαία και η μοντελοποίηση των κανόνων εφαρμογής των πολιτικών αυτών:

Όταν ένα γεγονός γίνεται αντιληπτό από τον χειριστή πολιτικής ενός υποκειμένου, αυτός ανατρέχει στο αποθετήριο πολιτικών ασφαλείας για να διαπιστώσει εάν προκύπτουν υποχρεώσεις για το υποκείμενο οι οποίες οφείλουν να εκκινήσουν. Εφόσον υπάρχουν θα ζητηθεί να σταλεί στον κατάλληλο στόχο η απαιτούμενη ενέργεια. Εάν υπάρχει πολιτική αποχής, τότε η ενέργεια θα απορριφθεί. Μόλις το υποκείμενο αιτηθεί την ενέργεια στον στόχο, αυτός θα πρέπει να επεξεργαστεί το αίτημα με βάση τις ισχύουσες πολιτικές ασφαλείας. Αυτό θα γίνει με τον ίδιο τρόπο, με ερώτηση στο αποθετήριο πολιτικών ασφαλείας. Εφόσον η ενέργεια επιτρέπεται θα πραγματοποιηθεί, διαφορετικά θα απορριφθεί. Η φορμαλιστική αντιμετώπιση του παραπάνω ακολουθεί:

Obligation / Refrain enforcement rule
$happens(requestAction(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_n)$ $\leftarrow holdsAt(oblig(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_m) \wedge T_m < T_n$
$happens(rejectAction(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_n)$ $\leftarrow holdsAt(refrain(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_m) \wedge T_m < T_n$
Access Control rule
$happens(doAction(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_n)$ $\leftarrow holdsAt(permit(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_m) \wedge T_m < T_n$

$$happens(rejectAction(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_n)\ )$$
$$\leftarrow holdsAt(deny(Subj, operation\ \ Targ, Action\ \ ParamList\ \ , T_m) \wedge T_m < T_n$$

Ο πρώτος κανόνας μοντελοποιεί την συμπεριφορά του χειριστή πολιτικής του υποκειμένου (έξυπνος agent) προκαλώντας το γεγονός του αιτήματος μιας ενέργειας όποτε μια υποχρέωση που αναφέρει αυτή την ενέργεια επαληθεύεται.

Ο δεύτερος κανόνας μοντελοποιεί την πολιτική ασφαλείας στο υποκείμενο με βάση την οποία πρέπει να απορριφθεί η ζητηθείσα ενέργεια εφαρμόζοντας κανόνα αποχής.

Ο τρίτος κανόνας μοντελοποιεί την πολιτική πρόσβασης στον στόχο και προκαλεί ένα γεγονός (*doAction*) εφόσον αυτό επιτρέπεται. Με τον τρόπο αυτό αλλάζει η κατάσταση του συστήματος.

Όμοια ο τέταρτος κανόνας απορρίπτει την εκτέλεση μιας ζητηθείσας ενέργειας.

Για να είναι δυνατή η αλληλεπίδραση των μοντέλων που έχουν περιγραφεί προηγουμένως, είναι αναγκαίο κάθε κανόνας στην πολιτική ασφαλείας να ορίζει την κατάλληλη λειτουργία (permit, deny, oblig, refrain) για κάθε ένα από τα γεγονότα. Για παράδειγμα μια πολιτική ασφαλείας θετικής πρόσβασης πρέπει να ορίζει πως το $permit\ \ Subj, Operation$ ισχύει όταν το συμβαίνει το γεγονός $requestAction\ \ Subj, Operation$ και οι περιορισμοί εφαρμογής της πολιτικής ισχύουν επίσης. Επί πρόσθετα, το fluent $permit\ \ Subj, Operation$ πρέπει να πάψει να ισχύει μόλις η ενέργεια έχει πραγματοποιηθεί. Αυτό απαιτείται ώστε να είναι δυνατή η επαναξιολόγηση του κανόνα σε επόμενα αιτήματα προς ενέργεια:

posAuth

$$initiates(requestAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , permit\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ ) \wedge Constraint$$
$$terminates(doAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , permit\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ )$$

negAuth

$$initiates(requestAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , deny\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ ) \wedge Constraint$$
$$terminates(rejectAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , deny\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ )$$

oblig

$$initiates(systemEvent\ \ E\ \ , oblig\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ ) \wedge Constraint$$
$$terminates(doAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , oblig\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ )$$

refrain

$$initiates(systemEvent\ \ E\ \ , refrain\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ , T_m)$$
$$\leftarrow isValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ \ ParamList\ \ ) \wedge Constraint$$

$terminates(rejectAction\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ ParamList\ \ , refrain\ \ Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ ParamList\ \ , T,$
$\leftarrow ValidSpec(Obj_{Subj}, operation\ \ Obj_{Targ}, Action\ ParamList\ \ )$

Για κάθε κανόνα οι όροι Obj_{Subj}, Obj_{Targ}, Action και Constraint μπορούν να απεικονιστούν ένας προς ένα τόσο σε PROLOG όπως είναι προφανές άλλωστε από τις παραπάνω δηλώσεις τους σε προτασιακή λογική, όσο και στην γλώσσα Ponder, καθώς οι περιορισμοί στην Ponder εκφράζονται με την Object Constraint Language[34]. Οι περιορισμοί της μορφής *Constraint* εκφράζονται και αυτοί με μια σειρά από συζεύξεις όρων *holdsAt*.

Ο κανόνας negAuth απεικονίζει μια πολιτική αρνητικής πρόσβασης ορίζοντας πως εάν το *Constraint* ισχύει και προκύπτει το γεγονός που ζητά την ενέργεια, τότε το αίτημα απορρίπτεται. Ο δεύτερος κανόνας δείχνει πως το fluent *deny* θα τερματιστεί μόλις η απόφαση για την απόρριψη της ενέργειας έχει παρθεί. Με αυτό τον τρόπο επιτυγχάνεται η δυνατότητα επαναχρησιμοποίησης του κανόνα σε επόμενα αιτήματα. Οι κανόνες τερματισμού για αυτούς τους περιορισμούς δεν έχουν ειδικότερους περιορισμούς άλλους εκτός από αυτούς που καθολικά ορίζει το σύστημα.

Ο κανόνας oblig ορίζει πως εάν το *Constraint* είναι αληθές όταν συμβεί το γεγονός, τότε η υποχρέωση του υποκειμένου να προβεί στην απαιτούμενη ενέργεια στο στόχο είναι ισχυρή. Όπως και με τον κανόνα posAuth, μια υποχρέωση τερματίζει όταν έχει γίνει η κλήση για την συγκεκριμένη ενέργεια. Εδώ υποκρύπτεται η υπόθεση πως η εκτέλεση της συγκεκριμένης ενέργειας είναι atomic operation και άρα θεωρείται πως έχει ολοκληρωθεί μετά το αίτημα.

Ο κανόνας της αποχής (refrain) ορίζει πως το fluent *refrain* τερματίζει μόλις αποφασιστεί πως η ζητούμενη ενέργεια δεν μπορεί να εκτελεστεί σύμφωνα με τα όσα ορίζει η πολιτική ασφαλείας.

Μια πλήρης ανάπτυξη πολιτικής ασφαλείας θα απαιτούσε την αρχικοποίηση των κανόνων initiates με συγκεκριμένα υποκείμενα, στόχους και λειτουργίες ορισμένες για το σύστημα υπό διαχείριση. Οι κανόνες από μόνοι τους απλά ορίζουν μια πολιτική και τις συνθήκες υπό τις οποίες είναι αληθείς, αλλά οι αρχικές συνθήκες του προβλήματος είναι απαραίτητες ώστε να περιγραφεί σωστά το πρόβλημα.

ΑΝΑΛΥΣΗ ΠΟΛΙΤΙΚΩΝ ΑΣΦΑΛΕΙΑΣ

Με δεδομένο πως οι περιγραφές των πολιτικών ασφαλείας που έχουν παρουσιαστεί μέχρι στιγμής έχουν τη δυνατότητα να περιγράψουν αντιφατικές καταστάσεις για τις ίδιες συνθήκες, συγκρούσεις μεταξύ επιμέρους πολιτικών μπορούν να ανακύψουν. Εμπειρικά και ειδικότερα κατά την υλοποίηση των πολιτικών ασφαλείας, όσο μεγαλύτερο το μέγεθος ενός συστήματος που καλύπτεται από αυτές και όσο πιο πολύπλοκες οι εξαρτήσεις των υποσυστημάτων του μεταξύ τους αλλά και με τον «έξω» κόσμο, τόσο βεβαιότερο είναι πως τέτοιες συγκρούσεις θα εμφανιστούν. Είναι χρήσιμο στον διαχειριστή ασφάλειας να τις γνωρίζει όσο το δυνατόν ταχύτερα, ώστε να μπορεί να αποφασίσει πριν καν την υλοποίηση της πολιτικής πώς θα

διαχειριστεί τις συγκρούσεις αυτές. Διαφορετικά κινδυνεύει να τρέχει πίσω από τα γεγονότα (παραβιάσεις ή άλλες ανεπιθύμητες ενέργειες που το σύστημα επέτρεψε να συμβούν).

Συγκρούσεις τροπικότητας που περιλαμβάνουν πολιτικές αδειοδότησης προκύπτουν όταν υπάρχουν δύο πολιτικές, μία που να επιτρέπει και μία που να απαγορεύει για την ίδια τριάδα υποκείμενου, στόχου και ενέργειας. Έστω πως ορίζεται το fluent authConflict το οποίο μπορεί να παρακολουθεί τέτοιες συγκρούσεις:

$$holdsAt\ authConflict\ Subj, Op\ , T_m \leftarrow holdsAt\ permit\ Subj, Op\ , T_m\ \wedge holdsAt\ deny\ Subj, Op\ , T_m$$

Με τον ίδιο τρόπο είναι δυνατός ο εντοπισμός συγκρούσεων μεταξύ υποχρεώσεων και αποχών:

$$holdsAt\ obligConflict\ Subj, Op\ , T_m \leftarrow holdsAt\ oblig\ Subj, Op\ , T_m\ \wedge holdsAt\ refrain\ Subj, Op\ , T_m$$

$$holdsAt(unauthObligConflict\ Subj, Op\ , T_m) \leftarrow holdsAt(oblig\ Subj, Op\ , T_m) \wedge holdsAt(deny\ Subj, Op\ , T_m)$$

Για λόγους συντομίας στην έκφραση η παράμετρος *Op* εκφράζει ό,τι το κατηγόρημα *operation* στις προηγούμενες εκφράσεις.

Όπως περιγράφεται στο [35] αυτές οι συγκρούσεις μπορεί να είναι συγκρούσεις καθήκοντος, ενδιαφέροντος, πολλαπλών διαχειριστών, προτεραιοτήτων στον ανταγωνισμό για πόρους του συστήματος και συγκρούσεις αυτοδιαχείρισης. Με βάση τα παραπάνω είναι δυνατό να οριστούν κανόνες και fluent που να μπορούν να αναγνωρίσουν τις συγκρούσεις αυτές.

Από τις πλέον κοινότυπες συγκρούσεις που εμφανίζονται κατά την κατάστρωση πολιτικών ασφαλείας είναι οι συγκρούσεις καθηκόντων. Αυτές εμφανίζονται όταν στο ίδιο υποκείμενο επιτρέπεται να εκτελέσει λειτουργίες που στο πλαίσιο της εφαρμογής ενδιαφέροντος είναι συγκρουόμενες. Για παράδειγμα στο σύστημα οικονομικής διαχείρισης μιας εταιρίας, η λειτουργία της εισαγωγής μιας πληρωμής και η λειτουργία της αποδοχής έκδοσης του εντάλματος πληρωμής είναι πιθανά αντικρουόμενες όταν πραγματοποιούνται από το ίδιο άτομο.

Οι κανόνες που ορίζονται για τον εντοπισμό τέτοιων συγκρούσεων, πρέπει να χρησιμοποιούν περιορισμούς που εξαρτώνται άμεσα από τις εφαρμογές και τα δεδομένα τους. Πριν όμως από τον ορισμό τέτοιων περιορισμών είναι χρήσιμο να ληφθούν υπόψη οι διαφορετικοί τύποι τους όπως έχουν μελετηθεί στο [36]:

- Μια σύγκρουση εμφανίζεται όταν το ίδιο υποκείμενο εκτελεί και τις δύο εργασίες στον ίδιο στόχο, όπως στο παράδειγμα που έχει ήδη αναφερθεί με τα εντάλματα πληρωμής.
- Σύγκρουση ενδιαφέροντος εμφανίζεται όταν το ίδιο υποκείμενο εκτελεί λειτουργίες σε πολλούς στόχους. Για παράδειγμα όταν μια τράπεζα παρέχει επενδυτικές συμβουλές σε πελάτη ενώ ταυτόχρονα ενεργεί ενάντια στο συμφέρον του (π.χ. πραγματοποιώντας συγχωνεύσεις με έναν ανταγωνιστή του).

- Διαφορετικά υποκείμενα εκτελούν κάθε μία από τις λειτουργίες στον ίδιο στόχο και το αποτέλεσμα κάθε μιας είναι ασύμβατο με τα υπόλοιπα (π.χ. στέλνοντας ένα κείμενο προς εκτύπωση μέσω ενός Unix server ενώ ταυτόχρονα εκτελείται shutdown).

Για τον εντοπισμό τέτοιων συγκρούσεων επεκτείνεται η γλώσσα με την προσθήκη του κατηγορήματος $conflictingOps\ conflictType,\ Ops$. Το *conflictType* συμβολίζει μια τιμή από τις *{conflictDuty, conflictInterest, conflictGoal, conflictSelfMgmt}*. Η λίστα *[Ops]* αποτελείται από όρους της μορφής *operation* όπως αυτοί έχουν χρησιμοποιηθεί προηγουμένως. Χρησιμοποιώντας το παράδειγμα των ενταλμάτων πληρωμής:

$$conflictingOps\ conflictDuty,\ operation\ payment, request\ paymentID, amount\ \ , operation\ payment, approve\ paymentID, amount$$

Ο διαχωρισμός καθηκόντων / αρμοδιοτήτων μπορεί να γίνει με πολλούς τρόπους. Μπορεί να είναι στατικός, οπότε και επιτυγχάνεται με το να μην επιτρέπεται σε κάποιο ρόλο στο σύστημα να εκτελέσει κάποια λειτουργία εφόσον του έχει δοθεί άδεια να εκτελεί άλλη λειτουργία για την οποία το σύστημα ήδη γνωρίζει μέσω του *conflictingOps* πως είναι συγκρουόμενες.

Υπάρχει όμως και ο δυναμικός διαχωρισμός αρμοδιοτήτων στον οποίο απαιτείται από το runtime εκτέλεσης του συστήματος να μην επιτρέψει την πραγματοποίηση αντικρουόμενων ενεργειών. Με τη μέθοδο του «Κινέζικου Τείχους» [37]. Σύμφωνα με αυτή την τεχνική ένα υποκείμενο δεν μπορεί να εκτελέσει αντικρουόμενες εντολές σε ένα στόχο εφόσον του έχει ήδη δοθεί άδεια να κάνει το ίδιο σε άλλο στόχο.

Σύγκρουση πολλαπλών διαχειριστών προκύπτει όταν διαφορετικά υποκείμενα (διαχειριστές) προσπαθούν να εκτελέσουν λειτουργίες σε ένα στόχο με ασύμβατο μεταξύ τους αποτέλεσμα. Χρησιμοποιώντας το παράδειγμα του εκτυπωτή:

$$conflictOfGoalsOps(conflictGoal, [operation\ \ printer, printDoc), operation(printer, shutdown)$$

Η δήλωση στο σύστημα γίνεται τότε:

$$\begin{aligned} &holdsAt\ \ conflictOfMultMgmt\ \ Subj_1, \dots, Subj_n\ \ , Ops\ \ , T_m) \\ &\quad \leftarrow holdsAt\ \ permit\ \ Subj_1, Op_1\ \ , T_1\ \ \wedge \dots \wedge holdsAt\ \ permit\ \ Subj_n, Op_n\ \ , T_n \\ &\quad \wedge conflictngOps\ \ conflictGoal, Ops\ \ \wedge memberOf\ \ Op_1, Ops\ \ \wedge \dots \\ &\quad \wedge memberOf\ \ Op_n, Ops\ \ \wedge T_1 \leq T_2 \leq \dots \leq T_n \end{aligned}$$

Με τον ίδιο τρόπο, χρησιμοποιώντας συζεύξεις από *holdsAt* και *memberOf* δηλώνονται και οι υπόλοιπες συγκρούσεις που μπορούν να προσδιοριστούν.

Χρησιμοποιώντας τα fluent σύγκρουσης που έχουν οριστεί ως τελικοί στόχοι στο λογικό πρόγραμμα, είναι δυνατό να ερωτηθεί το σύστημα για αλληλουχίες γεγονότων οι οποίες οδηγούν σε συγκρούσεις. Εφόσον δεν βρεθεί κάποια τέτοια σειρά γεγονότων, τότε η πολιτική ασφαλείας δεν πάσχει από συγκρούσεις αυτής της μορφής.

Η προσέγγιση του [12] που παρουσιάζεται μέχρι στιγμής ισχυρίζεται πως χρησιμοποιεί τις τεχνικές του [38] για την απαγωγική διαδικασία ελέγχου των πολιτικών ασφαλείας. Στο [38] αναφέρεται όμως πως «περιορισμοί στον αριθμό των σελίδων καθιστούν απαγορευτική την παρουσίαση του απαγωγικού εργαλείου. Έχει όμως υλοποιηθεί σε PROLOG χρησιμοποιώντας μια απλοποιημένη εκδοχή του [39]»:

Χρησιμοποιώντας τα fluent ως ιδιότητες ασφαλείας (safety) του συστήματος η συγκεκριμένη τεχνική μειώνει τη χρονική πολυπλοκότητα της απαγωγής κατά δύο χρονικές μονάδες.

Επιπλέον είναι δυνατό να δειχτεί πως το ερώτημα θα μπορεί πάντα να παράγει μια πλήρη εξήγηση για τις συγκρούσεις.

Ένας απαγωγικός planner σε Event Calculus

Δυστυχώς το [39] δεν είναι γενικά διαθέσιμο προς μελέτη και έτσι για να είναι δυνατή η κατανόηση του συστήματος που πραγματοποιεί το abductive reasoning χρειάζεται λίγο reverse engineering. Το demo query που φαίνεται στο [12] είναι της μορφής:

```
demo([holdsAt(obligConflict(printMgr,operation(printer,printDoc)), t1], [], Plan).
```

Για κάποιον που έχει μια μικρή έστω εμπειρία στην σχετική βιβλιογραφία των abductive planners γραμμένων σε PROLOG η χρήση του κατηγορήματος demo παραπέμπει στον planner που περιγράφεται στο [10]. Ακολουθεί μια προσπάθεια να γίνει κατανοητός ο συγκεκριμένος planner στον μέσο προγραμματιστή που δεν ασχολείται με την αποδεικτική διαδικασία πληρότητας (soundness and completeness) της συγκεκριμένης εργασίας. Αυτή η προσπάθεια δεν είναι καθόλου εύκολη όπως φαίνεται και από το [40] καθώς δέκα χρόνια μετά ούτε ο ίδιος ο συγγραφέας θυμόταν λεπτομέρειες εκτέλεσης του προγράμματος.

Η εργασία στο [10] έχει σαν έμπνευσή της το [42]. Η διαφορά τους είναι πως το [42] έχει στηριχτεί στην situation calculus ([8], [43]) ενώ το [10] στην Event Calculus με τη χρήση circumscription.

Το planning μπορεί να θεωρηθεί ως το αντίστροφο της χρονικής προβολής. Η χρονική προβολή στην Event Calculus μπορεί να αποτυπωθεί άμεσα σαν deduction. Αυτό με τη σειρά του σημαίνει πως το planning στην Event Calculus θεωρείται abductive task. Χρησιμοποιώντας την situation calculus, ένα πλάνο βρίσκεται με τη χρήση της συνάρτησης *Result* η οποία απεικονίζει μια ενέργεια και μια κατάσταση σε μια νέα κατάσταση. Η συνάρτηση αυτή όμως δεν μπορεί να διαχειριστεί *αφηγήσεις* (αφηγήσεις είναι συζεύξεις προτάσεων της μορφής *Happens(a,t)* και $t_1 < t_2$) των οποίων η διάταξη δεν είναι πλήρως γνωστή. Στον αντίποδα, η Event Calculus είναι (λόγω abduction) ο κατάλληλος υποψήφιος για μια τέτοια εργασία.

Έστω λοιπόν ένα κόσμος (domain) που αποτελείται από δύο ενέργειες και τρία fluent. Ο όρος Go(x) ορίζει την ενέργεια «πηγαίνω στο x» και ο όρος Buy(x) ορίζει την αντίστοιχη ενέργεια. Τα τρία fluent ορίζονται ως At(x) που ισχύει εφόσον ο παίκτης είναι στη θέση x, το Have(x) ισχύει όταν ο παίκτης έχει το x, και το Sells(x,y) όταν το μαγαζί x πουλάει το y. Έστω οι ακόλουθες αρχικές συνθήκες:

$$Initiates\ Go\ x\ ,At\ x\ ,t$$
$$Terminates\ Go\ x\ ,At\ y\ ,t\ \leftarrow x \neq y$$
$$Initiates\ Buy\ x\ ,Have\ x\ ,t\ \leftarrow HoldsAt\ At\ y\ ,t\ \wedge HoldsAt\ Sells\ y,x\ ,t$$
$$Initially_P\ Sells\ DIYShop, Drill$$
$$Initially_P\ Sells\ Supemarket, Banana$$
$$Initially_P(Sells\ Supermarket, Milk\ )$$

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Σημείωση: Σε σχέση με τον φορμαλισμό που χρησιμοποιήθηκε στο προηγούμενο τμήμα της εργασίας είναι προφανές πως $initiallyTrue \equiv Initially_P$και $initiallyFalse \equiv Initially_N$.

Μπορεί να πάρει κάποιος μια μπανάνα, γάλα και ένα τρυπάνι;

$$HoldsAt(Have\ \ Banana\ \ ,T) \wedge HoldsAt(Have\ \ Milk\ \ ,T) \wedge HoldsAt(Have\ \ Drill\ \ ,T)$$

Έστω μία Χημική Βιομηχανία. Μπορεί να είναι ασφαλής;

$$HoldsAt\ \ PlantSafe,t\ \ \leftarrow HoldsAt\ \ TankEmpty,t\ \ \wedge HoldsAt\ \ TemperatureLow,t$$
$$Initiates\ \ DrainTank,TankEmpty,t\ \ \leftarrow HoldsAt\ \ PressureNormal,t$$
$$Initiates\ \ CoolTank,TemperatureLow,t$$
$$HoldsAt(PressureNormal,t) \leftarrow HoldsAt(ValveOpen,t)$$
$$HoldsAt(PressureNormal,t) \leftarrow HoldsAt(BoilerOff,t)$$
$$Initiates(OpenValve,ValveOpen,t)$$
$$Initiates(TurnOffBoiler,BoilerOff,t)$$

Με αρχικές συνθήκες:

$$Initially_N\ \ PlantSafe$$
$$Initially_N\ \ TemperatureLow$$
$$Initially_N\ \ ValveOpen$$
$$Initially_N\ \ BoilerOff$$
$$Initially_N\ \ TankEmpty$$
$$Initially_N(PressureNormal)$$

Υπάρχει ασφαλές πλάνο;

Για τις ανάγκες υλοποίησης ενός προγράμματος PROLOG το οποίο θα μπορεί να δέχεται σαν είσοδο περιγραφές σε Event Calculus και θα δίνει ως έξοδο πλάνα που θα οδηγούν σε τελικές καταστάσεις χρησιμοποιώντας απαγωγική λογική θα χρησιμοποιηθεί η τεχνική του meta-interpreter. Ο πλέον απλός meta-interpreter γραμμένος σε PROLOG είναι ο ακόλουθος:

```
demo([]).
demo([G|Gs1]) :- axiom(G,Gs2),append(Gs2,Gs1,Gs3),demo(Gs3).
demo([not(G)|Gs]) :- not demo([G]),demo(Gs).
```

Το τρικ είναι να γίνουν compile οι προτάσεις object-level στο μετα-επίπεδο. Έτσι λοιπόν ο ίδιος meta-interpreter μπορεί να εκφραστεί ως:

```
demo([λ0|Gs1]) :- axiom(λ1,Gs2),append(Gs2,[λ2,…,λn|Gs1],Gs3),demo(Gs3).
```

Η συμπεριφορά του interpreter αυτού είναι ίδια με αυτή του αρχικού. Με τον ίδιο τρόπο μπορούν να μετασχηματιστούν και τα αξιώματα της Event Calculus και να γίνουν κομμάτι του μετα-επιπέδου.

Η εκτέλεση του μετα-επιπέδου της PROLOG, δεν ακολουθεί ακριβώς αυτή που ορίζεται και το object-level. Αυτό συμβαίνει επειδή στο μετα-επίπεδο υπάρχει ένας ακόμα βαθμός ελέγχου χάρη στον οποίο η σειρά με επιλύονται οι στόχοι (sub-goals).

Για την αναπαράσταση του αξιώματος
$!HoldsAt\ f,t_3 \leftarrow Happens\ a,t_1,t_2 \wedge Terminates\ a,f,t_1 \wedge t_2 < t_3 \wedge !Declipped(t_1,f,t_3)$ το οποίο δεν είναι σε πρόταση Horn, χρησιμοποιούμε το κατηγόρημα *neg*. Με τη χρήση του είναι δυνατή η αντικατάσταση του $!HoldsAt(f,t)$ με το *holds_at(neg(F),T)*. Με τη χρήση του *neg* διατηρείται κάπως ο φορμαλισμός της Event Calculus και η δυνατότητα να χειρίζεται ελλιπή πληροφορία για fluent που ισχύουν. Αυτή η δυνατότητα θα είχε χαθεί με τη χρήση της negation-as-failure.

Ο ρόλος του απαγωγικού meta-interpreter είναι να κατασκευάσει ένα υπόλοιπο από απαγωγικούς όρους (literals) οι οποίοι δεν μπορούν να αποδειχθούν από το πρόγραμμα. Στην περίπτωση της Event Calculus αυτά είναι τα *happens* και *before*. Έτσι ένας abductive meta-interpreter προκύπτει:

```
abdemo([],R,R).
abdemo([G|Gs],R1,R2) :- abducible(G), abdemo(Gs,[G|R1],R2).
abdemo([G|Gs1],R1,R2) :- axiom(G,Gs2),append(Gs2,Gs1,Gs3),abdemo(Gs3,R1,R2).
```

Οι όροι που είναι απαγώγιμοι ορίζονται με το κατηγόρημα *abducible/1*. Σύμφωνα με το [40] οφείλει να υπάρχει τουλάχιστον ένας τέτοιος στόχος πριν την εκτέλεση για την ανεύρεση ενός πλάνου.

Με την είσοδο του negation-as-failure αυξάνονται οι δυσκολίες. Αυτό συμβαίνει γιατί όπως αυξάνεται το υπόλοιπο, στόχοι που είχαν αποδειχτεί προηγουμένως μπορεί πλέον να μην ισχύουν. Επομένως οι συγκεκριμένοι στόχοι πρέπει να σημειωθούν για επανέλεγχο.

Στο πλαίσιο της μη πλήρους πληροφορίας δεν είναι δυνατό να χρησιμοποιήσουμε το αποτέλεσμα του negation-as-failure για υποθέσουμε την ολοκλήρωση κάποιου κατηγορήματος. Επομένως για αυτά τα κατηγορήματα χρειάζεται ειδικός χειρισμός. Στην παρούσα περίπτωση δεν υπάρχει πλήρης πληροφορία για το *before/2* επιτρέποντας έτσι μερικές διατάξεις αφηγήσεων να απεικονιστούν. Έτσι όταν ο meta-interpreter συναντά ένα αναιρούμενο στόχο *before* και συναντάει όταν πρέπει να εξηγήσει ένα αναιρούμενο στόχο *clipped* επιδιώκει να το αποδείξει. Ένας τρόπος να το πετύχει αυτό είναι να προσθέσει τον όρο στο υπόλοιπο αφού πρώτα ελέγξει εάν το νέο υπόλοιπο είναι συνεπές.

Το κατηγόρημα *before* δεν είναι η μόνη πηγή παροχής μη πλήρους πληροφορίας. Αντίστοιχα ζητήματα εγείρονται και από το *holds_at* το οποίο και κληρονομεί τη συμπεριφορά του από το before.

Ο πλήρης κώδικας του planner υπάρχει στο [44] . Για να λυθεί ένα πρόβλημα, χρειάζεται να περιγραφούν απευθείας στην PROLOG τα αποτελέσματα των ενεργειών με τη χρήση *initiates*, *terminates* και *releases*. Οφείλει να υπάρχει τουλάχιστον ένας στόχος abducible (στις δηλώσεις των προβλημάτων ορίζεται το *abducible(dummy)*). Οι ενέργειες ορίζονται μέσω του κατηγορήματος *executable/1*. Έτσι τοποθετείται το ερώτημα στο σύστημα χρησιμοποιώντας το κατηγόρημα *abdemo/2* στο οποίο το πρώτο όρισμα είναι μια σειρά από *holds_at* ενώ το δεύτερο περιέχει την απάντηση του ερωτήματος.

Μια προσέγγιση αντικειμενοστρεφούς προγραμματισμού

Από την παρουσίαση που έχει γίνει μέχρι τώρα είναι φανερή η δύναμη της χρήσης του λογικού προγραμματισμού για την επίλυση των ερωτημάτων που προκύπτουν από την κατασκευή των πολιτικών ασφαλείας. Οι συγγραφείς του [12] κάνουν το λάθος να καταφεύγουν στη χρήση του λογικού προγραμματισμού μεταφράζοντας απευθείας από το αντικειμενοστρεφές μοντέλο που χρησιμοποιούν. Αυτό έχει σαν αποτέλεσμα την παραγωγή λογικών εκφράσεων που είναι μεγάλες, δύσχρηστες και πολύπλοκες, καθιστώντας την πιθανή ανάγκη debugging δύσκολη.

Το πρόβλημα αυτό γίνεται φανερό ήδη από το γεγονός πως χρησιμοποιούν ένα αντικειμενοστρεφές μοντέλο προγραμματισμού χωρίς να το έχουν αντιληφθεί. Υπάρχουν τουλάχιστον τρία απλά μοντέλα αντικειμενοστρεφούς προγραμματισμού ([41], [46] και [47]) που θα μπορούσαν να έχουν χρησιμοποιηθεί, αντίθετα όμως χρησιμοποιούν ένα στο οποίο ακόμα και η απλούστερη διάσχιση της σχηματιζόμενης ιεραρχίας είναι δύσκολη και απαιτεί επιπλέον κατηγορήματα.

Υπάρχει όμως τρόπος να επιλυθεί το συγκεκριμένο πρόβλημα και να απλοποιηθούν οι χρησιμοποιούμενες εκφράσεις ελέγχου. Να χρησιμοποιηθεί η αντικειμενοστραφής γλώσσα προγραμματισμού Logtalk [48]. Η Logtalk είναι μια γλώσσα προγραμματισμού η οποία προσφέρει δυνατότητες μετα-προγραμματισμού, πολλαπλής κληρονομικότητας, προγραμματισμού βασισμένου σε γεγονότα (event driven programming), στοιχεία δηλαδή απαραίτητα όπως έχει φανεί από την μέχρι τώρα παρουσία για την υλοποίηση ενός αξιόπιστου συστήματος ελέγχου πολιτικών ασφαλείας με κατηγορήματα του Λογισμού Γεγονότων.

Σε σχέση με τα όσα έχουν παρουσιαστεί υπάρχει ακόμα ένα σημαντικό ζητούμενο: Βελτιστοποίηση της πολιτικής ασφαλείας και ανάλυσή της μέχρι τις τελικές συσκευές. Η διαδικασία δηλαδή της απόδοσης ενός στόχου από την διεύθυνση στα χαμηλότερα στρώματα της ιεραρχίας. Εάν η διαδικασία βελτιστοποίησης της πολιτικής ασφαλείας είναι εσφαλμένη, το αποτέλεσμα είναι αρνητικό σε όλα τα επίπεδα του οργανισμού. Σε ό,τι αφορά τον οργανισμό η διαδικασία αυτή απαιτεί τον ανθρώπινο παράγοντα και την εμπειρία του αντικειμένου. Σε ό,τι αφορά το συστημικό επίπεδο η διαδικασία αυτή μπορεί να είναι μερικώς αυτοματοποιημένη. Το ζητούμενο από μια διαδικασία βελτιστοποίησης είναι:

- Ορθότητα: Μια βελτιστοποίηση είναι ορθή όταν η σύζευξη όλων των πολιτικών που την αποτελούν δίνει τα ίδια αποτελέσματα με την αρχική πολιτική ασφαλείας.
- Συνέπεια: Μια βελτιστοποίηση είναι συνεπής όταν δεν υπάρχουν συγκρούσεις στο νέο σύνολο πολιτικών ασφαλείας
- Ελαχιστοποίηση: Μια βελτιστοποίηση είναι η ελάχιστη που μπορεί να επιτευχθεί εάν αφαιρώντας από το σύνολό της μία οποιαδήποτε πολιτική, το σύνολο που προκύπτει δεν είναι ορθό.

Εφαρμογή της Συλλογιστικής Κοινής Λογικής σε Πολιτικές Ασφαλείας με τη Χρήση Λογισμού Γεγονότων

Με τη χρήση του Λογισμού Γεγονότων είναι δυνατό να επιτευχθούν βελτιστοποιήσεις, όχι όμως με την υλοποίηση που έχει χρησιμοποιηθεί από το [12], δηλαδή την [44] όπως περιγράφεται στο [10]. Για το λόγο αυτό προτείνονται τα ακόλουθα βήματα που ομοιάζουν με αυτά της κατασκευής του [45] ώστε να γραφεί σε Logtalk ο κατάλληλος planner ο οποίος μπορεί να αντιμετωπίσει τέτοια ζητήματα. Ο λόγος που συμβαίνει αυτό είναι πως ο planner στο [10] βρίσκει ένα μόνο από τα πιθανά σενάρια που μπορεί να απαντούν το ερώτημα που γίνεται στο σύστημα. Δεν τα βρίσκει όλα (εάν υπάρχουν περισσότερα από ένα) άρα δεν βρίσκει και το βέλτιστο, παρόλο που ο φορμαλισμός το επιτρέπει.

Τα προτεινόμενα βήματα λοιπόν είναι:

Βήμα 1ο: Ανάγνωση του satisfiability suggested format

Με στόχο την κατασκευή ενός Event Calculus Reasoner ο οποίος θα κωδικοποιεί τα αξιώματά της σε προτάσεις των οποίων το ερώτημα θα ικανοποιεί το SAT[49], θα πρέπει να κατασκευαστεί αναγνώστης εκφράσεων του Satisfiability Suggested Format[50] καθώς θα πρέπει να μπορεί να χρησιμοποιηθεί εξωτερικός solver ο οποίος επικοινωνεί με το υπόλοιπο σύστημα με τη χρήση αυτού του format. Το συγκεκριμένο υποσύστημα θα κάνει χρήση των Definite Clause Grammars της PROLOG.

Βήμα 2ο: Επίλυση SAT

Οι διαθέσιμες επιλογές είναι αρκετές. Η επίλυση μπορεί να γίνει με τη χρήση του RelSat[51], του WalkSat[52] ή του MinSat[53]. Για την επικοινωνία του συστήματος με τα προγράμματα αυτά είναι απαραίτητη η λειτουργία του αναγνώστη του προηγούμενου βήματος.

Εναλλακτικά μπορεί να γίνει είτε υλοποίηση του WalkSAT σε PROLOG, είτε να χρησιμοποιηθεί το [54] που υλοποιεί ήδη σε PROLOG τον αλγόριθμο DPLL[55].

Βήμα 3ο: Κωδικοποίηση της Event Calculus στο SAT

Με τη χρήση των τεχνικών που αναφέρονται στο [9] και στο [56] θα επιτευχθεί η κωδικοποίηση των αξιωμάτων της Event Calculus και των ερωτημάτων που γίνονται μέσω αυτής σε προτάσεις του SAT των οποίων η ικανοποιησιμότητα θα αναζητηθεί από το σύστημα. Όλες οι εναλλακτικές θα παρουσιαστούν.

Βήμα 4ο: Χρήση αντικειμένων της Logtalk

Καθώς κάθε πολιτική ασφαλείας απεικονίζει την ιεραρχία ενός οργανισμού είναι φανερό πως απαιτείται η χρήση μιας αντικειμενοστραφούς γλώσσας για τον έλεγχο και τη βελτίωση των υφιστάμενων πολιτικών. Οι συγγραφείς του [12] χρησιμοποιώντας τη γλώσσα Ponder και την OCL, επιχείρησαν να υπερκεράσουν το πρόβλημα της μη εύκολης ανεύρεσης προγραμματιστών σε περιβάλλοντα λογικού προγραμματισμού, θυσιάζοντας και επανυλοποιώντας (βλέπε και 10ος νόμος του Greenspun) αναγκαίο κομμάτι της λογικής για τη διαλειτουργικοτητα. Αυτός είναι και ο λόγος που ενώ έχουν αποδείξει την ισχύ του μαθηματικού μοντέλου, όταν χρειάστηκε να δημιουργήσουν μηχανισμούς ελέγχου των περιγραφόμενων

πολιτικών, αυτοί είναι δύσχρηστοι, αντιαισθητικοί και πρακτικά ανεφάρμοστοι και από τον πλέον έμπειρο προγραμματιστή σε λογικά περιβάλλοντα.

Η Logtalk μπορεί να καλύψει το κενό αυτό και με βάση όλα τα παραπάνω είναι δυνατό να επιτευχθεί τόσο η αποτύπωση της ιεραρχίας ενός οργανισμού σε ένα σύστημα PROLOG με έν φυσικό και εύχρηστο τρόπο, όσο και η υλοποίηση ενός Event Calculus Planner να είναι τέτοια που να εξυπηρετεί και τις ανάγκες βελτιστοποίησης των εφαρμοζόμενων πολιτικών.

Με δεδομένο πως με την τεχνική JPL [57] είναι δυνατή η συνεργασία ενός συστήματος PROLOG με άλλα προγράμματα γραμμένα σε Java, είναι δυνατό να χρησιμοποιηθούν τα καλύτερα από δύο κόσμους: Να αποφευχθούν οι συνέπειες του εμπειρικού νόμου του Greenspun και να χρησιμοποιηθούν οι πλούσιες βιβλιοθήκες της Java για την κατασκευή user interface που θα μπορεί να κάνει τη καθημερινή εργασία ευκολότερη τόσο στον διαχειριστή συστημάτων, όσο και στον υπεύθυνο ασφαλείας.

Συμπεράσματα

Στην μεταπτυχιακή αυτή διατριβή μελετήθηκε η δύναμη του μοντέλου του Λογικού Προγραμματισμού όπως αυτό συνοδεύεται από τον φορμαλισμό του Λογισμού Γεγονότων με σκοπό την αποτύπωση, τον έλεγχο και πιθανά τη βελτίωση των πολιτικών ασφαλείας που εφαρμόζονται σε ένα σύστημα. Η μοντελοποίηση ενός συστήματος οποιασδήποτε ιεραρχίας είναι δυνατή με αυτά τα εργαλεία και είναι δυνατός ο εντοπισμός (και η απάλειψη) συγκεκριμένων συγκρούσεων που μπορεί να προκύψουν από την κατάρτιση της πολιτικής ασφαλείας.

Δόθηκε επίσης μια σύντομη περιγραφή του πλέον δημοφιλούς meta-interpreter που υλοποιεί την Event Calculus σε PROLOG και εντοπίστηκαν δύο λεπτά και μη καταγεγραμμένα μέχρι σήμερα στοιχεία της λειτουργίας του, μετά από προσωπική επικοινωνία με το δημιουργό.

Τέλος, προτάθηκαν τα βήματα για την ανάπτυξη από την αρχή ενός συστήματος Event Calculus που θα μπορεί να υποστηρίζει οποιαδήποτε ιεραρχία και θα έχει τις εξής ιδιαιτερότητες: Θα κωδικοποιεί τα ερωτήματα για την Event Calculus σε ισοδύναμα προβλήματα του SAT και δεύτερον επειδή θα είναι γραμμένο σε Logtalk (μία αντικειμενοστρεφή γλώσσα γραμμένη σε PROLOG) θα μπορεί να έχει απλούστερες περιγραφές των πολιτικών ασφαλείας και των απαιτούμενων ελέγχων από το υποτυπώδες και δύσχρηστο σύστημα που έχουν κατασκευάσει οι συγγραφείς της Ponder. Επιπλέον, με τη δυνατότητα που προσφέρει η σύνδεση PROLOG και Java μέσω του JPL, είναι δυνατή η κατασκευή διεπαφών τα οποία θα απλοποιούν το έργο του διαχειριστή και θα μπορούν ακόμα (χάρη στη δύναμη απεικόνισης του μοντέλου που προσφέρει η Event Calculus) να χρησιμοποιηθούν και ως εργαλεία στρατηγικής.

Βιβλιογραφία

1. Systems Thinking. (n.d.). In Wikipedia. Retrieved April 20, 2011, from http://en.wikipedia.org/wiki/Systems_thinking
2. Cobit, C. S. C. (2000). COBIT Framework. IT Governance Institute.
3. ISO/IEC 27001:2005. Information Technology – Security Techniques – Information security management systems – Requirements.
4. Greenspun's Tenth Rule. (n.d.). In Wikipedia. Retrieved April 20, 2011 from http://en.wikipedia.org/wiki/Greenspun's_Tenth_Rule
5. Allen, J. (n.d.). Whither Software Engineering. Unpublished manuscript. School of Engineering, Santa Clara University
6. Blackburn, P., Bos, J., & Striegnitz, K. (2006). Learn Prolog Now! College Publications.
7. Kowalski, R., & Sergot, M. (1985). A logic based calculus of events. (C. Thanos & J. W. Schmidt, Eds.)New Generation Computing, 4(1), 67-95. Springer-Verlag.
8. Shanahan, M. (1997). Solving the Frame Problem (p. 407). MIT Press.
9. Mueller, E. T. (2006). Commonsense Reasoning. Morgan Kaufman.
10. Shanahan, M. (2000). An abductive event calculus planner. Journal of Logic Programming, 44(1-3), 207–240. Elsevier
11. Fluent . (n.d.). In Wikipedia. Retrieved, April 20, 2011, from http://en.wikipedia13.org/wiki/Fluent_(artificial_intelligence)
12. Bandara, A., Lupu, E. C., & Russo, A. (2003). Using Event Calculus to Formalise Policy Specification and Analysis. Proceedings POLICY 2003 IEEE 4th International Workshop on Policies for Distributed Systems and Networks, (June), 26-39. IEEE Computer Society.
13. McCarthy, J. (1980). Circumscription - A Form of Non-Monotonic Reasoning.Artificial Intelligence.

14. McCarthy, J. (1986). Applications of Circumscription to Formalizing Common Sense Knowledge. Artificial Intelligence, 28(1), 89-116
15. Mueller, E. T. (n.d.). Event calculus axioms (quick reference). Retrieved April 20, 2011, from http://decreasoner.sourceforge.net/csr/ecaxioms.pdf
16. Miller, R., & Shanahan, M. (2002). Some alternative formulations of the event calculus. Notes (Vol. 2408, pp. 452-490). Springer Berlin/ Heidelberg.
17. Mueller, E. T. (2004). Event Calculus Reasoning Through Satisfiability. Journal of Logic and Computation, 14(5), 703-730. Oxford University Press.
18. Shanahan, M. (1999). The ramification problem in the event calculus. IJCAI99 Proceedings of the 16th international joint conference on Artifical intelligence,16, 140-146. Morgan Kaufmann Publishers Inc.
19. Miller, R., & Shanahan, M. (1999). The event calculus in classical logic - alternative axiomatisations.
20. Closed World Assumption. (n.d.). In Wikipedia. Retrieved April 20, 2011, from http://en.wikipedia.org/wiki/Closed_world_assumption
21. Reiter, R. (1978). On Closed World Data Bases. In H. Gallaire & J. Minker (Eds.), Logic and Data Bases (pp. 55-76). Plenum Press.
22. Jiang, W. S., & Wee, W. G. (1987). Commonsense reasoning in Prolog. Proceedings of the 15th annual conference on Computer Science CSC 87, 138-143. ACM Press.
23. Herbrand Structure. (n.d.). In Wikipedia. Retrieved April 20, 2011, from http://en.wikipedia.org/wiki/Herbrand_structure
24. Negation as failure. (n.d.). In Wikipedia. Retrieved April 20, 2011, from http://en.wikipedia.org/wiki/Negation_as_failure
25. Abductive Reasoning. (n.d.). In Wikipedia. Retrieved April 20, 2011, from http://en.wikipedia.org/wiki/Abductive_reasoning
26. Hanks, S., & McDermott, D. (1986). Default Reasoning, Nonmonotonic Logics and the Frame Problem. Proceedings Of The National Conference On Artificial Intelligence (pp. 328-333). Morgan Kaufmann.
27. Kakas, A. C., Kowalski, R. A., & Toni, F. (1998). The Role of Abduction in Logic Programming. In C. J. H. D M Gabbay & J. A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming 5 (pp. 235-324). Oxford University Press.
28. Shanahan, M. (2000). An abductive event calculus planner. Journal of Logic Programming, 44(1-3), 207–240. Elsevier.
29. Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The Ponder Policy Specification Language. Policy, 1995, 18-38. Springer-Verlag.
30. Ponder2 Wiki. (2010). Retrived April 20, 2011, from http://www.ponder2.net
31. Ranum, M. (1997). Thinking about firewalls V2.0: Beyond perimeter security.Information Security Technical Report, 2(3), 33-45.
32. Warren, K. (2010). Strategy Dynamics Essentials. Strategy Dynamics Limited.
33. System Dynamics Society. (2011). Retrieved April 20, 2011, from http://www.systemdynamics.org/

34. Beckert, B., Keller, U., & Schmitt, P. (2002). Translating the object constraint language into first-order predicate logic. VERIFY 2002 1st International Verification Workshop.
35. Lupu, E. C., & Sloman, M. (1999). Conflicts in policy-based distributed systems management. IEEE Transactions on Software Engineering, 25(6), 852-869. IEEE Press.
36. Moffett, J. D., & Sloman, M. S. (1994). Policy Conflict Analysis in Distributed System Management. Journal of Organizational Computing, 4(1), 1-22.
37. Brewer, D. F. C., & Nash, M. J. (1989). The Chinese Wall Security Policy.Proceedings 1989 IEEE Symposium on Security and Privacy, pp, Issue:(May), 206-214. IEEE Comput. Soc. Press.
38. Russo, A., Miller, R., Nuseibeh, B., & Kramer, J. (2002). An abductive approach for analysing event-based requirements specifications. Lecture Notes in Computer Science, 0. Springer
39. Kakas, A. C., & Michael, A. (1995). Integrating Abductive and Constraint Logic Programming. Proceedings of the Twelfth International Conference on Logic Programming ICLP95 (pp. 399-417). MIT Press.
40. (Shanahan, M. personal communication, April 15, 2011)
41. O' Keefe, R. (1990). The Craft of Prolog (p. xix + 387). The MIT Press.
42. Green, C. (1969). Application of Theorem Proving to Problem Solving. In B. Webber & N. Nilsson (Eds.), Readings in Artificial Intelligence (pp. 219-239). Tioga.
43. Situation Calculus. (n.d.). In Wikipedia. Retrieved April 20, 2011, from, http://en.wikipedia.org/wiki/Situation_calculus
44. Shanahan, M. (n.d.). Event Calculus Planner source code. Retrieved from http://www-ics.ee.ic.ac.uk/~mpsha/CogRob/planning/planner42.txt
45. Mueller, E. T. (n.d.). Commonsense Reasoning with the Discrete Event Calculus Reasoner. Retrieved from http://decreasoner.sourceforge.net/
46. Exploring Prolog. (1993). Retrieved April 20, 2011, from http://www.amzi.com/articles/prolog_fun.htm
47. Bratko, I. (2001). Prolog Programming for Artificial Intelligence. (A. D. McGettrick, Ed.)Book (Vol. 64, p. 352). Addison-Wesley.
48. Moura, P. (2003). Logtalk -- Design of an Object-Oriented Logic Programming Language.
49. Boolean Statisfiability Problem. (n.d.). In Wikipedia. Retrieved, April 20, 2011, from http://en.wikipedia.org/wiki/Boolean_satisfiability_problem
50. Challenge, D. (1993). Satisfiability: Suggested Format. DIMACS Challenge. DIMACS.
51. Bayardo, R. RelSat. Retrieved April 20, 2011, from http://code.google.com/p/relsat/
52. Selman, B., Kautz, H., & Cohen, B. (1996). Local search strategies for satisfiability testing. (M. Trick & D. S. Johnson, Eds.)Proceedings of the Second DIMACS Challange on Cliques Coloring and Satisfiability, 00, 26:521--532. AMS.
53. MinSAT. http://minsat.se/ Retrieved April 20, 2011.

54. Howe, J. M., King, A. (2010). A Pearl on SAT Solving in Prolog. (Blume, M. and Vidal, G. Eds)Tenth International Symposium on Functional and Logic Programming, Lecture Notes in Computer Science, page 10. Springer-Verlag
55. DPLL algorithm. (n.d.). In Wikipedia. Retrieved, April 20, 2011, from http://en.wikipedia.org/wiki/DPLL_algorithm
56. Mueller, E. T. (2004). Event Calculus Reasoning Through Satisfiability. Journal of Logic and Computation, 14(5), 703-730. Oxford University Press.
57. JPL – A Java Interface to Prolog. Retrieved from http://www.swi-prolog.org/packages/jpl/java_api/index.html