ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση συστημάτων κρίσιμης ασφάλειας

Διπλωματική Εργασία της

Κατσιαούνη Νομικής (ΑΕΜ: 1093)

Επιβλέπων Καθηγητής: Κατσαρός Παναγιώτης

ΘΕΣΣΑΛΟΝΙΚΗ
Σεπτέμβριος 2008

Πρόλογος

Η τεχνολογική εξέλιξη έχει ως συνέπεια την αυξανόμενη πολυπλοκότητα στη σύλληψη και στην ανάλυση των συστημάτων με γενικό τρόπο. Η κοινωνική ανάπτυξη συγχωρεί όλο και σε μικρότερο βαθμό τα απρόσμενα γεγονότα και ακόμη λιγότερο τα ατυχήματα τα οποία προέρχονται από συστήματα που χρησιμοποιούνται σε όλο και περισσότερου τομείς και με όλο και πιο σημαντικές συχνότητες.

Στόχος της διπλωματικής αυτής είναι η ανάπτυξη και μοντελοποίηση του μηχανισμού ελέγχου ασφαλείας σε ένα σιδηροδρομικό δίκτυο, όπου χρέος του είναι να παρεμποδίσει τρένα από συγκρούσεις και εκτροχιασμούς επιτρέποντας συγχρόνως την κίνησή τους στο δίκτυο. Η γλώσσα που χρησιμοποιήθηκε για τη μοντελοποίηση του συστήματος είναι η AltaRica και τα εργαλεία εφαρμογής της είναι το Cecilia Ocas. Επίσης, στόχος μας είναι η παρουσίαση κάποιων γενικών στοιχείων όσον αφορά στο model-checking (έλεγχος μοντέλων) και την ασφάλεια συστημάτων, τη γλώσσα AltaRica και το εργαλείο που χρησιμοποιήθηκε ώστε να παράγουμε το τελικό αποτέλεσμα. Τέλος , αναλύουμε τον τρόπο σχεδίασης του μοντέλου μας, τη συλλογική πορεία που ακολουθήσαμε έως ότου να καταλήξουμε στο τελικό προϊόν, και τις λειτουργίες του και προτείνουμε κάποιες επεκτάσεις που θα μπορούσαν να υλοποιηθούν.

Θα ήθελα να ευχαριστήσω θερμά τον κύριο Κατσαρό Παναγιώτη για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου τη συγκεκριμένη διπλωματική καθώς και για την βοήθεια και την υποστήριξη καθ’ όλη τη διάρκεια εκπόνησής της όπως επίσης και τον Μώκο Κωνσταντίνο και τον Μπασαγιάννη Στυλιανό οι οποίοι που προσέφεραν την πολύτιμη βοήθεία τους για την υλοποίηση της εργασίας και ήταν κοντά μου από την αρχή έως και το τέλος της. Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου που με στήριζαν καθ’ όλη τη διάρκεια της.

Νομική Κατσιαούνη

6 Σεπτεμβρίου 2008

Περιεχόμενα

1 Εισαγωγή

Στο παρελθόν, ο μηχανισμός εσωτερικού ελέγχου αναπτύχθηκε και μελετήθηκε κυρίως ως μια λειτουργία του ελέγχου ενός σιδηροδρομικού δικτύου, όπου το καθήκον του ήταν να παρεμποδίζει τα τρένα από τις συγκρούσεις και τους εκτροχιασμούς, ενώ ταυτόχρονα να επιτρέπει τις κινήσεις τους [4]. Η άποψή μας είναι ότι ο μηχανισμός εσωτερικού ελέγχου είναι ένα μέσο για την αποκλειστικά συγχρονισμένη πρόσβαση σε διανεμημένους πόρους δικτύου και η εφαρμογή του επεκτείνεται πέρα από αυτήν του ελέγχου σιδηροδρομικού δικτύου. Σε όλο και περισσότερα βιομηχανικά προϊόντα (αεροσκάφη, αυτοκίνητα, πυρηνικά εμφυτεύματα κτλ.), τα συστήματα υλικού και λογισμικού παίζουν συνεχώς όλο και πιο σημαντικό ρόλο, ιδιαίτερα στην υλοποίηση λειτουργιών ελέγχου. Αυτά τα συστήματα είναι κρίσιμα για τη ασφάλεια (safety-critical) και πρέπει να είμαστε προσεχτικοί στο σχεδιασμό και στον εξαντλητικό έλεγχο. Ένα χαρακτηριστικό των συστημάτων αυτών είναι ότι απαιτούν αρκετούς τύπους συστατικών : υλικά (μηχανικά, ηλεκτρικά, χημικά) και προγραμματιστικά, αλληλεπιδρώντας όλα αυτά με ποικίλους τρόπους (συγχρονισμούς, ανταλλαγές μηνυμάτων, αισθητήρες ανίχνευσης και ελεγκτές, κτλ.). Συνεπώς, ένα μαθηματικό μοντέλο αυτών των συστημάτων πρέπει να είναι ικανό να περιγράψει τα ξεχωριστά συστατικά καθώς και τον τρόπο που αλληλεπιδρούν, και να προσδιορίσει το αποτέλεσμα της αλληλεπίδρασης [15].

Η διαδικασία επιβολής ασφάλειας είναι αναπόσπαστο τμήμα της διαδικασίας ανάπτυξης ενός συστήματος. Η διαδικασία επιβολής ασφάλειας περιλαμβάνει αναγνώριση των απαιτήσεων και επικύρωση.

Ο έλεγχος για την ασφάλεια ενός συστήματος παρεμποδίζει κάποιους χειρισμούς από το να συμβούν, εκτός αν προκληθούν από κάποια γεγονότα. Οι μηχανισμοί εσωτερικού ελέγχου χρησιμοποιούνται σε οδικά συστήματα ελέγχου ασφαλείας(για παράδειγμα σε συστήματα εσωτερικού ελέγχου για σιδηροδρομικά δίκτυα), καθώς επίσης χρησιμοποιούνται και σε τομείς εφαρμογών όπως ηλεκτρικά κυκλώματα, πρωτόκολλα επικοινωνίας και ψηφιακοί ελεγκτές. Οι μηχανισμοί εσωτερικού ελέγχου πρέπει να δι-

ασφαλίζουν σίγουρη ασφάλεια και ορθότητα και το γεγονός αυτό καθιστά τη μοντελοποίηση ως την πιο σημαντική υπόθεση στο σχεδιασμό.

Τα συστήματα ελέγχου απαιτείται να εμφανίζουν απόλυτη ασφάλεια. Στις περισσότερες περιπτώσεις τα συστήματα αυτά παρουσιάζουν ένα σημαντικό βαθμό πολυπλοκότητας πράγμα που καθιστά δύσκολη την ανάλυσή τους, λόγω του μεγάλου χώρου καταστάσεων των αποτελεσμάτων. Επιπλέον, η εξαντλητική προσομοίωση και οι δοκιμές μπορούν να μην εντοπίσουν κάποια λάθη του συστήματος. Έτσι, η ανάπτυξη σύνθετων και κρίσιμων για την ασφάλεια συστημάτων απαιτεί τη χρησιμοποίηση τυπικών μεθόδων και εργαλείων για το σχεδιασμό και τις προδιαγραφές του συστήματος.

Η συγκεκριμένη εργασία αφορά στη υλοποίηση ενός κρίσιμου για την ασφάλεια συστήματος το οποίο επιτρέπει την ορθή κίνηση τρένων σε ένα σιδηροδρομικό δίκτυο. Στο σημείο αυτό πρέπει να αναφέρουμε ότι το έργο βασίστηκε στην εργασία του Μπασαγιάννη Στυλιανού “Fail-safe network interlocking control by Distributed Signal Boxes”[5]. Στην εργασία του παρουσιάζει τη μοντελοποίηση ενός σιδηροδρομικού δικτύου το οποίο αποτελείται από σταθμούς και DSBs τα οποία ελέγχουν την κίνηση των τρένων στο δίκτυο. Η μοντελοποίηση στηρίχθηκε στο εργαλείο SPIN. Η προσφορά μας τώρα είναι ότι εμπλουτίσαμε το μοντέλο εφαρμόζοντας την τεχνική του fail-safe για τη λειτουργία του δικτύου. Δηλαδή, προσθέσαμε για κάθε σταθμό και DSB ένα “ανίτγραφό” τους το οποίο ενεργοποιείται μόλις αποτύχει ένας σταθμός ή ένα DSB και συνεχίζει την ορθή λειτουργία του δικτύου. Επίσης, προσθέσαμε την οντότητα του supervisor η οποία είναι υπεύθυνη για την εναλλαγή ρόλων στους σταθμούς και στα DSBs όπως επίσης και για την επαναφορά κάποιας οντότητας σε περίπτωση που υποστεί βλάβη. Η μοντελοποίηση έγινε στη γλώσσα AltaRica με τη βοήθεια του εργαλείου Cecilia Ocas.

Όσον αφορά στη δομή τής συγκεκριμένης εργασίας, στο δεύτερο κεφάλαιο παραθέτουμε κάποια στοιχεία για το model checking, τα εργαλεία του και αναλυτικότερα για το εργαλείο Cecilia Ocas το οποίο και χρησιμοποιήθηκε στη μοντελοποίηση. Στο τρίτο κεφάλαιο κάνουμε λόγο για τη γλώσσα μοντελοποίησης AltaRica στην οποία γράφτηκε το μοντέλο, στο τέταρτο αναλύουμε τη λογική και τον τρόπο ανάπτυξης του μοντέλου ενώ στο πέμπτο παραθέτουμε τα αποτελέσματά του και προτείνουμε κάποιες επεκτάσεις. Τέλος, στο παράρτημα παρουσιάζονται κάποιες εικόνες από το Cecilia Ocas καθώς και ο κώδικας για το μοντέλο.

2 Έλεγχος μοντέλων (Model Checking)

2.1 Εισαγωγή στο Model Checking

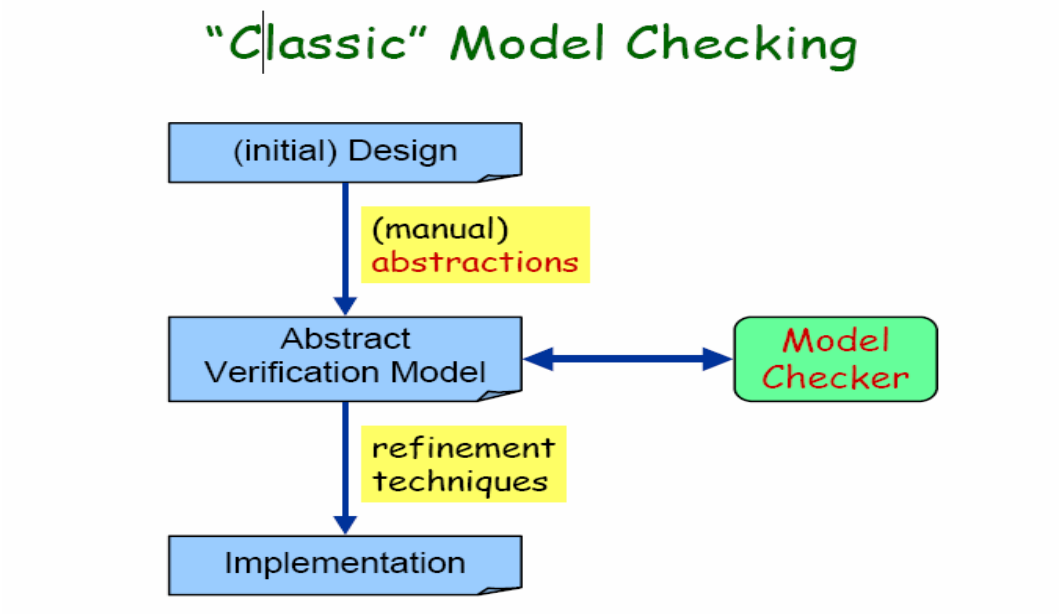
Η ανάπτυξη κρίσιμων συστημάτων συνοδεύεται με προσεχτικές αναλύσεις ασφάλειας. Ο σύλληψη και η ανάπτυξη των συστημάτων αυτών συνδέονται με οικονομικούς στόχους, όπως τη μείωση του κόστους και του χρόνου της ανάπτυξης, αλλά επίσης και το σεβασμό των κανόνων ασφάλειας. Ο κύκλος ανάπτυξης του συστήματος υποβάλλεται σε μια επιπρόσθετη διαδικασία επικύρωσης και επαλήθευσης. Οι διεργασίες αξιολόγησης που συνδέονται με την ασφάλεια λειτουργίας, όπου τοποθετούμε το επίπεδο της εμπιστοσύνης το οποίο αποδίδεται σε ένα σύστημα όταν αυτό χρησιμοποιείται σωστά, καταλαμβάνουν κυρίαρχη θέση. Το πρώτο βήμα είναι η ανάλυση των αναγκών που επιτρέπει στη συνέχεια τον προσδιορισμό των σημαντικών λειτουργιών που θέλουμε για το σύστημα. Αυτές οι λειτουργίες υψηλού επιπέδου μελετώνται μαζί με το ρίσκο που συνδέονται- αυτή η πρώτη ανάλυση της ασφάλειας της λειτουργίας καλείται FHA (Functional Hazard Analysis). Γενικά, οι ομάδες της ανάπτυξης και της ασφάλειας της λειτουργίας διαχωρίζονται, το γεγονός αυτό δίνει μια μείζουσα σημασία στους τρόπους αλληλεπίδρασης ανάμεσα σε αυτές ομάδες.

Η ανάλυση των ρίσκων παράγει στην έξοδο έναν αριθμό απαιτήσεων που θα πρέπει να ικανοποιήσει το σύστημα για να ανταπεξέλθει στους στόχους της ασφάλειας. Οι λειτουργίες έπειτα παράγονται, κρατώντας έναν απολογισμό των αποτελεσμάτων των απαιτήσεων από το FHA , για να προτείνουν μια λειτουργική απότμηση (ονομαζόμενη architecture fonctionnelle – λειτουργική αρχιτεκτονική) και μια αρχιτεκτονική για προκαταρκτική υποστήριξη (αποκαλούμενη architecture matérielle –υλιστική αρχιτεκτονική). Κατά τη διάρκεια του καθορισμού των λειτουργιών, μελέτες σε θέματα ασφάλειας διεξάγονται για να επικυρώσουν το σεβασμό των αποτελεσμάτων των απαιτήσεων του FHA πάνω στο θέμα της αποταμίευσης πόρων. Αυτό το στάδιο καλείται

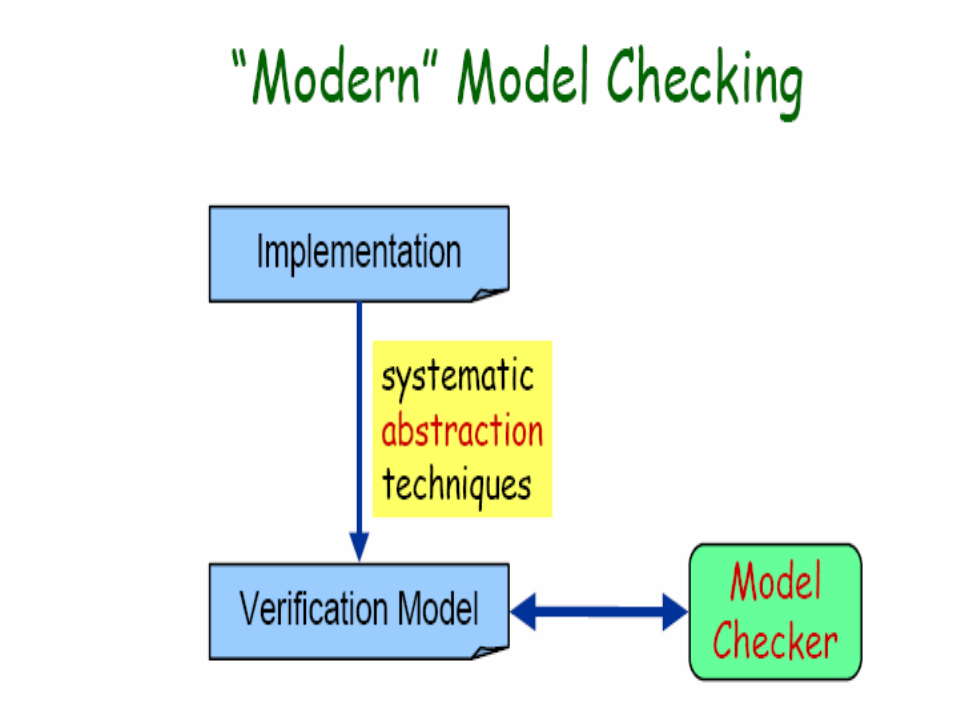
PSSA (Preliminary System Safety Assessment). Το να βρεθεί μια λύση μπορεί επομένως να γίνει μετά από κάποιον αριθμό επαναλήψεων ανάμεσα στις ομάδες ανάπτυξης και ασφάλειας της λειτουργίας. Μόλις αποκτηθεί συμφωνία των δύο μερών, η ακριβής ανάπτυξη αρχίζει. Ακολουθεί έπειτα η ενίσχυση του κύκλου με όλες τις απαραίτητες δοκιμές όσον αφορά στην εξαγωγή στην αγορά μαζί με ενισχύσεις εκ μέρους της ομάδας ασφάλειας της λειτουργίας κάτω από την ανάπτυξη του συστήματος SSA (System Safety Assessment)[10].

Η πιο διαδεδομένη γλώσσα μοντελοποίησης και προδιαγραφών για το model checking είναι η linear temporal logic (LTL). Κάθε φόρμουλα περιγράφει μία κατάσταση σε κάποιο σημείο του χρόνου. Πέρα από την LTL μπορούμε να χρησιμοποιήσουμε και second-order λογικές αν θέλουμε να υποστηρίξουμε διακλαδώσεις αντί του σειριακού χρονισμού, όπως η CTL (computation tree logic).

Στην εικόνα που ακολουθεί παρουσιάζεται η κλασική διαδικασία ελέγχου μοντέλων. Αρχικά, έχουμε τον αρχικό σχεδιασμό του συστήματος, στη συνέχεια εφαρμόζοντας αφαιρέσεις προκύπτει το αφαιρετικό μοντέλο επαλήθευσης το οποίο αλληλεπιδρά με τον ελεγκτή μοντέλου και τέλος εφαρμόζουμε τεχνικές εκλέπτυνσης και καταλήγουμε στην υλοποίηση.



Στην ακόλουθη εικόνα παρουσιάζεται το σύγχρονο Model Checking στο οποίο αρχικά έχουμε την υλοποίηση και στη συνέχεια εφαρμόζοντας συστηματικές τεχνικές αφαίρεσης παίρνουμε το μοντέλο πιστοποίησης το οποίο αλληλεπιδρά με τον ελεγκτή μοντέλου.



Υπάρχουν τέσσερεις βασικές τεχνικές για την επιβεβαίωση της ορθότητας του υ-λικού και λογισμικού συστημάτων:

- Προσομοίωση (Simulation)
- Δοκιμή (Testing)
- Συμπερασματική Επικύρωση (Deductive Verification)
- Έλεγχος Μοντέλων (Model Checking)

Επαλήθευση συστήματος μέσω model checking

Η διαδικασία του Model Checking:

- Μοντελοποίηση -modelling
- Προδιαγραφή -specification
- Επαλήθευση -verification

Μοντελοποίηση -Modelling

Τι είναι: είναι μια διαδικασία εξέτασης αν ένα μοντέλο ικανοποιεί μια δοθείσα φόρμουλα. Για τη μοντελοποίηση συστημάτων χρησιμοποιούμε πεπερασμένα αυτόματα [9].

Λόγω των περιορισμών σε χρόνο και μνήμη, η μοντελοποίηση ενός σχεδίου ίσως μπορεί να απαιτεί χρήση αφαίρεσης.

Χρησιμοποιούμε ένα τύπο γράφου μετάβασης κατάστασης επονομαζόμενο ως Kripke structure για να μοντελοποιήσουμε το σύστημα.

Τί είναι Kripke

Μία Kripke structure σε ένα σύνολο ατομικών προτάσεων.

ΑΠ είναι μία τετράδα M = (S, S0, R, L), όπου

- S είναι ένα πεπερασμένο σύνολο καταστάσεων.
- S0 ⊆ S είναι το σύνολο των αρχικών καταστάσεων.
- R ⊆ S × S είναι μία σχέση μετάβασης.
- L: S -> 2(ΑΠ) είναι μία συνάρτηση η οποία προσδιορίζει κάθε κατάσταση με το σύνολο των ατομικών προτάσεων αληθές σε αυτήν την κατάσταση.

Παράδειγμα με φούρνο μικροκυμάτων

Μοντελοποίηση με Kripke-Structure
M = (S, S0, R, L)

- S = (S1, S2, S3, S4)
- S1 είναι η αρχική κατάσταση
- R = ({S1, S2} {S2, S1}, {S1, S4}, {S4, S2}, {S2,S3}, {S3, S2}, {S3, S3}
- L (S1) = {¬close, ¬ start, ¬ cooking} L (S2) ={close, ¬ start, ¬ cooking} L (S3) = {close, start,cooking} L (S4) = {¬close, start, ¬ cooking}

Graph of the Kripke-Structure M of microwave-oven

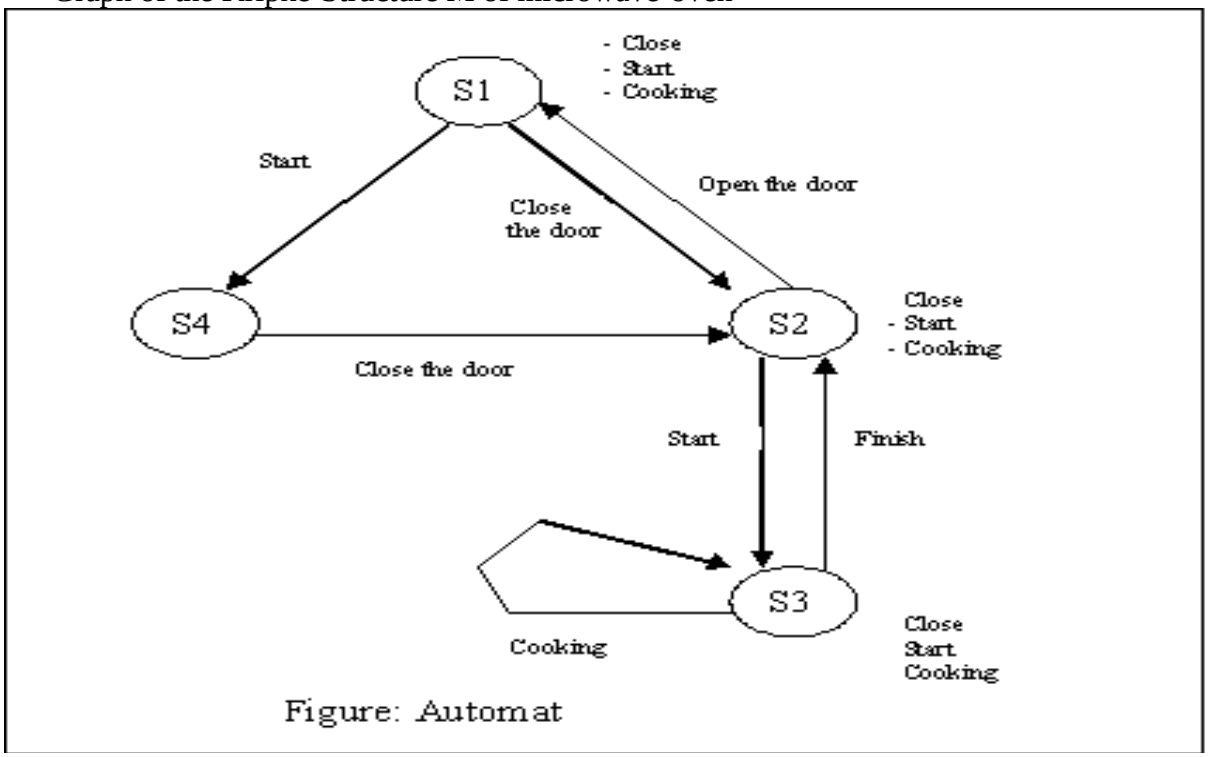


Figure: Automat

Προδιαγραφή -Specification

Τελεστές για τη Χρονολογική Λογική (Temporal Logic)
Πέντε βασικοί χρονικοί προσδιορισμοί
1. X (‘‘next time’’)
2. F (‘‘in the future’’)
3. G (‘‘globally’’)
4. U (‘‘until’’)
5. R (‘‘Release’’)

Δύο ποσοτικοποιητές για τη χρονολογική λογική
1. A (‘‘always’’)
2. E (‘‘exists’’)

Τρεις κύριοι τρόποι αναπαράστασης της Χρονολογικής Λογικής:

• CTL* (Computation Tree Logic*)

• CTL (Computation Tree Logic) με 10 τελεστές βάσης: AX και EX, AF και EF, AG και EG, AU και EU, AR και ER.

• LTL (Linear Temporal Logic)

*υπερσύνολο της CTL.

Παράδειγμα με φούρνο μικροκυμάτων

Specification με CTL-Formal

1. AG (start => AF cooking)

2. AG ((close ∧ start) => AF cooking)

Επαλήθευση –Verification

CTL* -Model-Checking

CTL -Model-Checking

LTL -Model-Checking

Παράδειγμα με φούρνο μικροκυμάτων (1)

Στην πρώτη CTL-Formal : AG (start => AF cooking)

1) Αλλαγή σε ¬EF (start ∧ EG ¬ cooking)

2) Από τις απλές φόρμουλες σε περισσότερο σύνθετες έως ότου όλες να είναι αληθείς.

• S (start) = {S3, S4}

• S (¬cooking) = {S1, S2, S4}

• S (EG ¬ cooking) = {S1, S2, S4}

• S (start ∧ EG ¬ cooking) = {S4}

• S (EF (start ∧ EG ¬ cooking)) = {S1, S2, S3, S4}

• S (¬ (EF (start ∧ EG ¬ cooking))) = {}

Παράδειγμα με φούρνο μικροκυμάτων (2)

Στη δεύτερη CTL-Formal: AG ((close ∧ start) => AF cooking)

1) Αλλαγή σε ¬ EF(close ∧ start ∧ EG ¬ cooking)

2) Τώρα ο αλγόριθμος μπορεί να εφαρμοστεί στη φόρμουλα

• S (close)= {S2, S3}

• S (start)= {S3, S4}

• S (¬ cooking) = {S1, S2, S4}

• S (EG ¬ cooking) = {S1, S2, S4}

• S (close ∧ start ∧ EG ¬ cooking) = {}

• S (EF (close ∧ start ∧ EG ¬ cooking) = {}

• S (¬ (EF (close ∧ start v EG ¬ cooking)) = {S1, S2, S3, S4}

2.2 Εργαλεία ελέγχου μοντέλων

Τα εργαλεία του model checking πιστοποιούν αυτόματα εάν ισχύει M |≓ ϕ, όπου M είναι ένα (πεπερασμένων καταστάσεων) μοντέλο συστήματος και η ιδιότητα (property) ϕ δηλώνεται με μια τυπική σημειογραφία.

Πρόβλημα: Έκρηξη στο χώρο καταστάσεων (state space explosion).

Αν και το μοντέλο είναι πεπερασμένων καταστάσεων παρουσιάζει εκθετική αύξηση.

Τα εργαλεία του model checking αντιμετωπίζουν το πρόβλημα της συνδυαστικής έκρηξης του χώρου καταστάσεων το οποίο πρέπει να διευθετηθεί με σκοπό την επίλυση προβλημάτων πραγματικού χρόνου. Υπάρχουν αρκετές προσεγγίσεις για την αντιμετώπιση του προβλήματος[16].

1. Οι συμβολικοί αλγόριθμοι αποφεύγουν τη δημιουργία γράφου για το FSM (finite state machine – είναι μοντέλο κάποιας συμπεριφοράς που συντίθεται από πεπερασμένο αριθμό καταστάσεων, μεταβάσεων ανάμεσα στις καταστάσεις και ενεργειών. Είναι αφηρημένο μοντέλο μιας μηχανής με στοιχειώδη εσωτερική μνήμη.), αντίθετα αναπαριστούν το γράφο χρησιμοποιώντας μία φόρμουλα σε προτασιακή λογική. Η χρήση δυαδικών διαγραμμάτων απόφασης (binary decision diagrams -BDDs) έγινε δημοφιλής από την εργασία του Ken McMillan (1992).
2. Οι αλγόριθμοι ελέγχου μοντέλων με περιορισμούς εκτυλίσσουν το FSM σε ένα αριθμό σταθερών βημάτων k και ελέγχουν εάν μία παραβίαση ιδιότητας μπορεί να συμβεί σε k ή περισσότερα βήματα. Αυτό τυπικά συνεπάγεται την κωδικοποίηση του μοντέλου με περιορισμούς ως ένα στιγμιότυπο τύπου SAT. Η διαδι-

κασία μπορεί να επαναληφθεί με όλο και μεγαλύτερες τιμές του k μέχρις ότου όλες οι πιθανές παραβάσεις να αποκλειστούν.
3. Μερική ελαχιστοποίηση κανόνων μπορεί να εφαρμοστεί.
4. Η αφαίρεση επιχειρεί να πιστοποιήσει ιδιότητες στο σύστημα απλοποιώντας το πρώτα. Το απλοποιημένο σύστημα συνήθως δεν ικανοποιεί τις ίδιες ιδιότητες με το αρχικό και έτσι κρίνεται απαραίτητη μία διαδικασία εκλέπτυνσης.

Τα εργαλεία του Model checking αρχικά αναπτύχθηκαν με κίνητρο τη λογική ορθότητα διακριτών καταστάσεων, αλλά σήμερα έχουν επεκταθεί και ασχολούνται με πραγματικού χρόνου τύπους υβριδικών συστημάτων.

Στη συνέχεια παρουσιάζουμε ενδεικτικά κάποια εργαλεία ελέγχου μοντέλων.

SPIN (= Simple Promela Interpreter): Το SPIN είναι ένα από τα πιο δυναμικά εργαλεία ελέγχου μοντέλου. Είναι ένα εργαλείο για την πιστοποίηση ορθότητας κατανεμημένου λογισμικού. Ασχολείται με την ανάλυση της λογικής συνέπειας παράλληλων συστημάτων, ιδιαιτέρα πρωτοκόλλων επικοινωνίας δεδομένων. Τα παράλληλα συστήματα περιγράφονται στη γλώσσα μοντελοποίησης PROMELA η οποία επιτρέπει τη δυναμική δημιουργία παράλληλων διεργασιών και η επικοινωνία λαμβάνει χώρα μέσω (σύγχρονων ή ασύγχρονων) καναλιών μηνυμάτων. Οι ιδιότητες που πρέπει να πιστοποιηθούν εκφράζονται σε Linear Temporal Logic (LTL) φόρμουλα. Επιπρόσθετα με το έλεγχο μοντέλων, το SPIN μπορεί να χρησιμοποιηθεί ως προσομοιωτής ακολουθώντας ένα πιθανό μονοπάτι εκτέλεσης του συστήματος και παρουσιάζοντας στο χρήστη τα αποτελέσματα της εκτέλεσης [11].

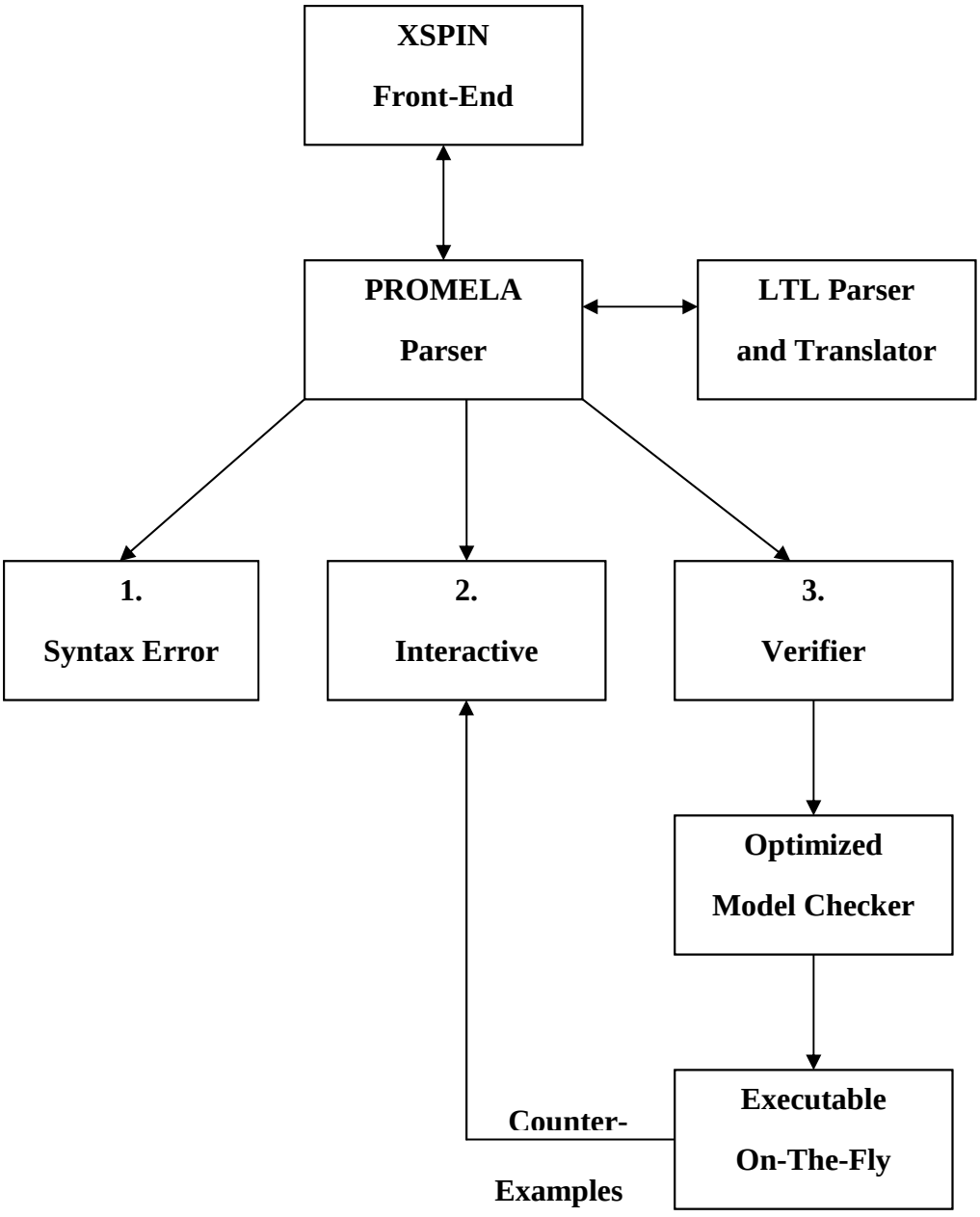
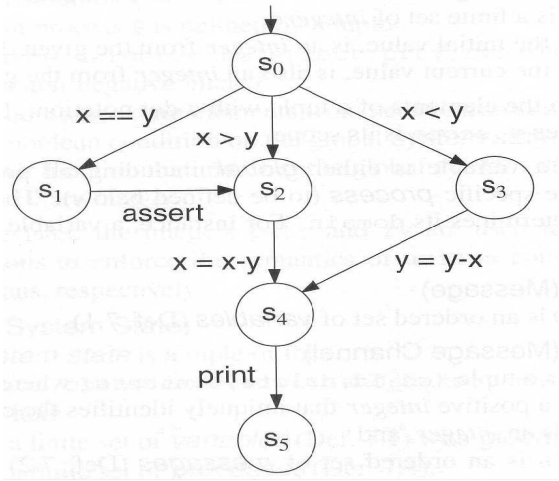


Fig.1. The structure of SPIN simulation and verification.

Process Meta Language.

```
active proctype not_euclid(int x, y)
{
if
:: x < y -> L: x = x-y;
:: x < y -> y = y-x;
:: x==y -> assert(x!=y); goto L
fi;
printf("%d\n", x);
}
```



KRONOS: είναι ένα εργαλείο που αναπτύχθηκε με στόχο την πιστοποίηση σύνθετων συστημάτων πραγματικού χρόνου –αυτά είναι συστήματα τα οποία πρέπει να επιτελέσουν μία λειτουργία μέσα σε αυστηρό χρονικό πλαίσιο. Στο Kronos τα συστατικά των συστημάτων μοντελοποιούνται από χρονικά ρυθμισμένα αυτόματα (αυτόματα που επεκτείνονται με ένα πεπερασμένο σύνολο ρολογιών -clocks, χρησιμοποιούμενα για να εκφράσουν χρονικούς περιορισμούς) και οι απαιτήσεις ορθότητας εκφράζονται σε πραγματικού χρόνου χρονολογική λογική (real-time temporal logic) TCTL. Ελέγχει εάν ένα αυτόματα με χρονικούς περιορισμούς ικανοποιεί μία TCTL- φόρμουλα. Το εργαλείο διανέμεται δωρεάν για ακαδημαϊκούς σκοπούς [16].

CHESS: είναι ένα αυτοματοποιημένο εργαλείο ανίχνευσης λαθών σε πολυνηματικό λογισμικό μέσω συστηματικής εξερεύνησης των προγραμμάτων. Ανιχνεύει λάθη όπως data-races, deadlocks, hangs, and data-corruption τα οποία είναι πολύ δύσκολο να βρεθούν με τα ισχύοντα εργαλεία ελέγχου. Μόλις το CHESS εντοπίσει λάθος παρέχει μία πλήρη επαναληπτική εκτέλεση του προγράμματος που οδήγησε στο λάθος συμβάλλοντας έτσι στη διαδικασία εκσφαλμάτωσης [16].

BOGOR: είναι ένα εκτατό λογισμικό ελέγχου μοντέλων με υψηλού επιπέδου αλγορίθμους ελέγχου μοντέλων, οπτικοποίηση, και διασύνδεση χρήστη σχεδιασμένη να υποστηρίζει συγχρόνως γενικού και ειδικού σκοπού προγράμματα ελέγχου μοντέλων. Αν και υπάρχουν αρκετοί διαθέσιμοι ελεγκτές μοντέλων, το Bogor παρέχει έναν αριθμό

καινοτομικών δυνατοτήτων, πράγμα που το κάνει ιδιαίτερα ευχάριστο και εξυπηρετικό στον έλεγχο ιδιοτήτων μιας ποικιλίας μοντέρνων προγραμμάτων, για την κατασκευή δικών σας ειδικού-σκοπού μηχανών, και για τη χρήση του με σκοπό τη διδασκαλία των αρχών του model checking [16].

Mec 5 Model – Checker : Το Mec 5 είναι ένας ελεγκτής μοντέλων. Η υλοποίησή του βασίζεται στη δομή δεδομένων "δυαδικά διαγράμματα απόφασης (binary decision diagrams – BDDs)"[6].

Είναι ένα εργαλείο το οποίο μας βοηθάει να διαχειριστούμε μοντέλα γραμμένα στη γλώσσα AltaRica. Μας επιτρέπει να ορίσουμε σχέσεις σε μια γλώσσα προδιαγραφών με βάση τις περιγραφές των μοντέλων. Η γλώσσα αυτή είναι πολύ εκφραστική και μας επιτρέπει να ορίσουμε σύνθετες σχέσεις ανάμεσα σε μοντέλα.

Προσδιορισμός της γλώσσας

Η γλώσσα που χρησιμοποιείται από το Mec 5 συμπίπτει ακριβώς με τη μ-ανάλυση του Park η οποία είναι first-order λογική που επεκτείνεται με fixpoints στις σχέσεις.

First-Order Logic

Οι προτάσεις δομούνται χρησιμοποιώντας τους λογικούς συνδέσμους ~ για άρνηση, & για σύζευξη, | για διάζευξη. Οι μεταβλητές μπορεί να είναι λογικές (bool), ακέραιοι σε διάστημα τιμών (π.χ. [0,10]) ή αλφαριθμητικά (π.χ. {on, off}).

Η εισαγωγή των First-order ποσοτικοποιητών της γλώσσας μάς επιτρέπει να εκφράσουμε υπαρξιακές και καθολικές φόρμες όπως "για όλους τους διαδόχους…" και "υπάρχει ένας διάδοχος ο οποίος…" των οποίων η σημασιολογία συνήθως δίνεται ρητά και οι οποίες είναι συνήθως ο μόνος σύνδεσμος ανάμεσα στη γλώσσα και το μοντέλο που μελετάται. Χρησιμοποιείται η ακόλουθη σύνταξη: ∃x.p γράφεται <x>p και ∀x.p γράφεται [x]p.

Fixpoints στις σχέσεις

Δοσμένης μιας μονοτονικής συνάρτησης στις σχέσεις είναι δυνατό να υπολογίσουμε την least (+=) ή τη greatest (-=) σχέση η οποία είναι ένα Fixpoint της συνάρτη-

σης. Για παράδειγμα, ο υπολογισμός της μεταβατικής κλειστότητας T μιας σχέσης R μπορεί να γραφτεί στο Mec 5 χρησιμοποιώντας ένα least fixpoint:

T(x, y) += R(x, y) | <z> (R(x, z) & T(z, y));

Υλοποίηση

Το Mec 5 χρησιμοποιεί Δυαδικά Διαγράμματα Απόφασης (BDDs) για να αναπαραστήσει τις σχέσεις. Αυτή η επιλογή αρμόζει στη γλώσσα μας , επειδή λογικοί όπως και First-order χειρισμοί είναι αποδοτικοί σε BDDs. Ο έλεγχος για ισότητα μπορεί να γίνει σε σταθερό χρόνο, πράγμα το οποίο γίνεται όταν υπολογίζουμε Fixpoints. Εκφράσεις είναι διαχειρίσιμες από διανύσματα των BDDs.

Ενσωμάτωση με την AltaRica

Η AltaRica βασίζεται σε αυτόματα με περιορισμούς και παρέχει ευκολίες στη μοντελοποίηση σύνθετων συστημάτων. Ένας κόμβος (node) της AltaRica μπορεί να οριστεί ιεραρχικά, πρώτα μοντελοποιώντας μικρά τμήματα του συστήματος (τους ίδιους τους κόμβους) και έπειτα συνενώνοντάς τους σε sub-nodes. Η εξ ορισμού σημασιολογία των sub-nodes είναι να τρέχουν ασύγχρονα, εκτός αν κάποια γεγονότα είναι ρητά συγχρονισμένα. Δύο μέσα επικοινωνίας παρέχονται:

Memory Sharing. Το τμήμα assertion σε έναν node μπορεί να συσχετίσει τοπικές μεταβλητές με μεταβλητές των sub-nodes του. Εξαναγκάζοντας δύο μεταβλητές να είναι ίσες, είναι εύκολο να μοιράσουμε πληροφορίες μεταξύ των κόμβων. Μπορούν να χρησιμοποιηθούν κα σύνθετοι περιορισμοί. (A.f1 = B.f) & (A.f2 < C.f)

Events Synchronization. Διανύσματα συγχρονισμού <A.a, B.b> καθορίζουν ποια γεγονότα θα συμβούν μαζί, εμποδίζοντάς τα από το να εμφανιστούν ανεξάρτητα, και επιτρέποντας σε μία μετάβαση να πυροδοτήσει όλα τα γεγονότα παράλληλα.

Παράδειγμα:

Δίνουμε ένα απλό παράδειγμα μοντέλου σε AltaRica το οποίο είναι ένας βρόχος από τον οποίο είναι δυνατό να δραπετεύσουμε μη-ντετερμινιστικά. Προσδιορίζουμε το προϊόν δύο τέτοιων βρόχων και έπειτα υπολογίζουμε το σύνολο των καταστάσεων από τις οποίες είναι αναπόφευκτο να οδηγηθούμε σε μια αδρανή κατάσταση.

NotEpsilon (e : main!ev) := ~ (e.S1. = “ ” & e.s2. = “ ”);

UnavDead(c) += [e][c’] ((main ! t (c, e, c’) & NotEpsilon(e)) => UnavDead(c’);

node LoopExit

state s : [0, 2];

event a, b;

trans s = 0 |- a -> s := 1;

s = 1 |- b -> s:= 0;

s = 1 |- b -> s := 2;

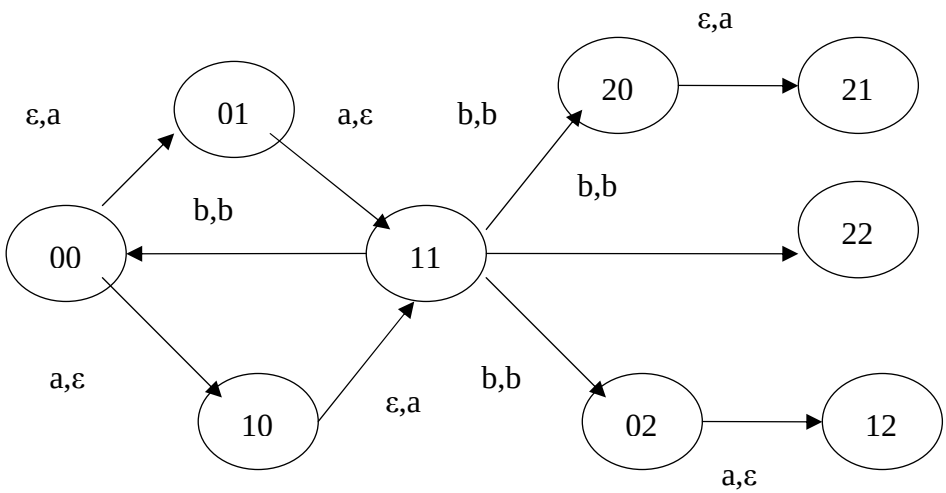
edon

node main

sub S1, S2 : LoopExit;

sync <S1.b, S2.b>;

edon



Σημειολογία του κόμβου main χωρίς ε βρόχους

({S1.s = 2, S2.s = 2})

({S1.s = 1, S2.s = 2})

({S1.s = 0, S2.s = 2})

({S1.s = 2, S2.s = 1})

({S1.s = 2, S2.s = 0})

Αποτέλεσμα

PRISM: είναι ένας πιθανοκρατικός ελεγκτής μοντέλων, ένα εργαλείο για μοντελοποίηση και ανάλυση συστημάτων τα οποία παρουσιάζουν τυχαία ή πιθανοκρατική συμπεριφορά. Υποστηρίζει τρεις τύπους πιθανοκρατικών μοντέλων: αλυσίδες Marcov διακριτού χρόνου (discrete-time Markov chains -DTMCs), αλυσίδες Marcov συνεχούς χρόνου (continuous-time Markov chains -CTMCs) και Μαρκοβιανές διαδικασίες απόφασης (Markov decision processes -MDPs), και επιπλέον επεκτάσεις αυτών των μοντέλων με κόστη και αμοιβές. Το PRISM χρησιμοποιείται για την ανάλυση συστημάτων ενός ευρέος φάσματος εφαρμογών συμπεριλαμβανομένων πρωτόκολλα πολυμέσων και επικοινωνίας, πρωτόκολλα ασφαλείας, βιολογικά συστήματα και πολλά άλλα. Τα μοντέλα περιγράφονται με τη γλώσσα PRISM (PRISM language), μία απλή, βασιζόμενη σε καταστάσεις γλώσσα. Παρέχει υποστήριξη για αυτοματοποιημένη ανάλυση μιας μεγάλης ποικιλίας ποσοτικών ιδιοτήτων των μοντέλων, όπως για παράδειγμα “ποια είναι η πιθανότητα μία βλάβη να προκαλέσει την κατάρρευση του συστήματος σε 4 ώρες;”, “ποιο είναι το αναμενόμενο μέγεθος της ουράς μηνυμάτων μετά από 30 λεπτά;”, “ποια είναι η χειρότερη περίπτωση για το χρόνο τερματισμού ενός αλγορίθμου;”. Η γλώσσα προσδιορισμού ιδιοτήτων (property specification language) ενσωματώνει τις χρονολογικές λογικές PCTL και CSL καθώς και επεκτάσεις για ποσοτικούς προσδιορισμούς και κόστη/αμοιβές. Ενσωματώνει συμβολικές δομές δεδομένων (symbolic data structures) και αλγορίθμους βασισμένους σε BDDs (Binary Decision Diagrams) και MTBDDs (Multi-Terminal Binary Decision Diagrams). Το PRISM διανέμεται δωρεάν και είναι ανοικτού κώδικα [16].

ALLOY ANALYSER: Alloy είναι μια δομημένη γλώσσα μοντελοποίησης που βασίζεται σε first-order λογική για να εκφράσει σύνθετους περιορισμούς και συμπεριφορές. Το Alloy Analyzer είναι ένα εργαλείο επίλυσης περιορισμών το οποίο παρέχει πλήρως

αυτοματοποιημένη προσομοίωση και έλεγχο. Έχει αναπτυχθεί από το Software Design Group του MIT. Τεχνικά μιλώντας, το Alloy Analyzer είναι ένας 'model finder'. Δοθείσας μιας λογικής φόρμουλας (σε Alloy) προσπαθεί να βρει ένα μοντέλο το οποίο κάνει τη φόρμουλα αληθή [16].

2.3 Cecilia Ocas

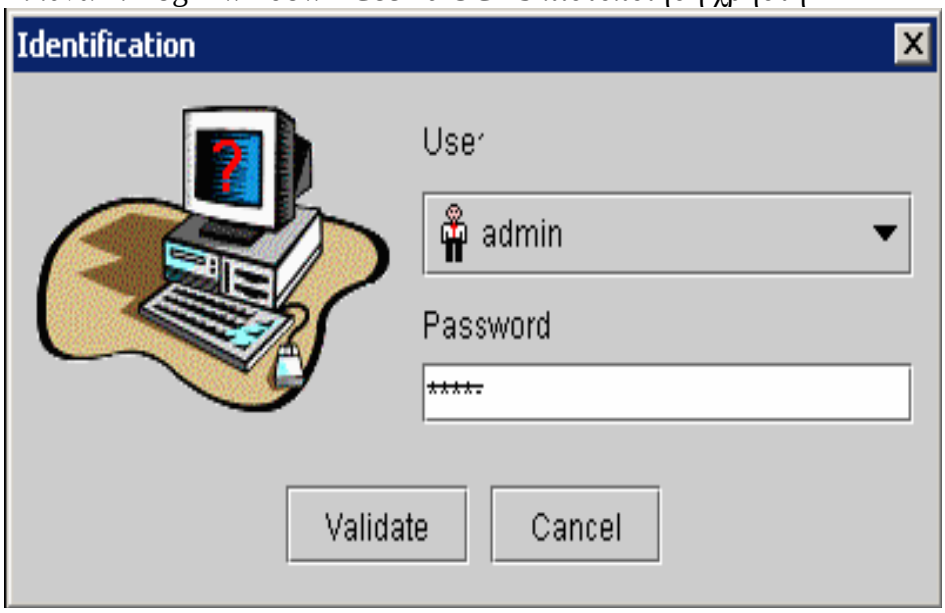
2.3.1 Διαδραστική γραφική προσομοίωση

Ένας μηχανικός ασφάλειας μπορεί να ελέγξει τα αποτελέσματα από την εμφάνιση κάποιας αποτυχίας (failure) σε μία αρχιτεκτονική συστήματος, χρησιμοποιώντας το Cecilia OCAS graphical interactive simulator. Ο μηχανικός ασφάλειας επιλέγει ένα γεγονός (event) και η κατάσταση των αποτελεσμάτων υπολογίζεται από τον προσομοιωτή. Αφού οι αποτυχίες (failures) είναι γεγονότα (events) σε ένα μοντέλο AltaRica, ο μηχανικός ασφάλειας μπορεί να προσθέσει στο μοντέλο έναν αριθμό από γεγονότα αποτυχιών (failure events) προκειμένου να διαπιστώσει εάν μία συνθήκη αποτυχίας γίνεται αντιληπτή (όπως για παράδειγμα μία βλάβη σε ένα ή περισσότερα κανάλια τάσης).

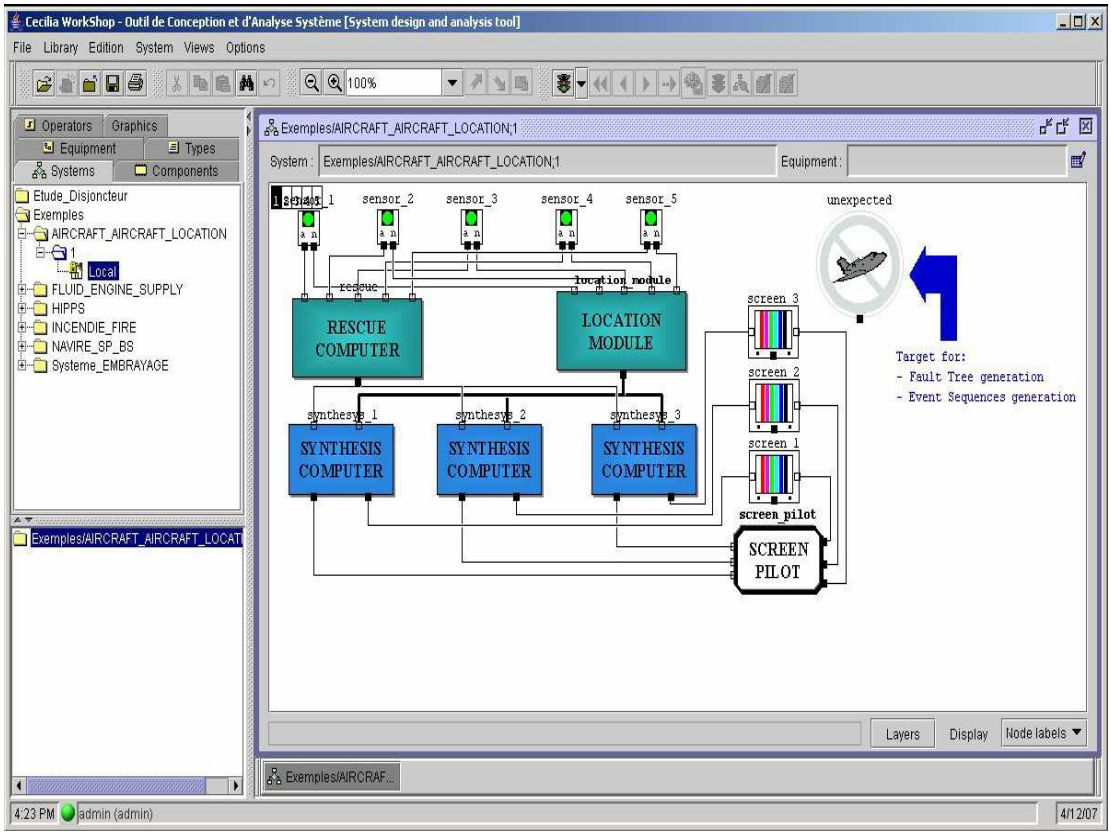
Πλεονεκτήματα: Αρκετά εικονίδια μπορούν να συνδεθούν με κάποιο συστατικό του μοντέλου αναλόγως με την κατάστασή του. Για παράδειγμα, ένα πράσινο κουτί εμφανίζεται εάν ο παρατηρητής λαμβάνει ενέργεια, ενώ ένα κόκκινο την αντίθετη περίπτωση. Αυτά τα εικονίδια συμβάλλουν σε μια γρήγορη αποτίμηση σε περίπτωση που εμφανιστεί κάποια συνθήκη αποτυχίας [1, 13].

Εικόνα 1: A Window display

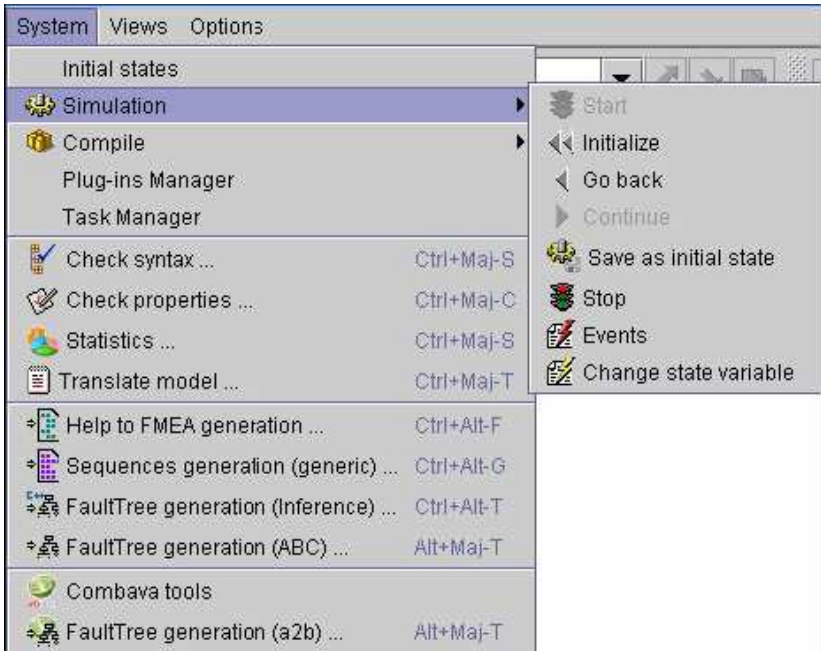
Εικόνα 2: Login window - Cecilia OCAS πιστοποίηση χρήστη



Εικόνα 3 : Περιβάλλον εργασίας



Εικόνα 4 : μενού του συστήματος. Από το μενού Simulation διαχειριζόμαστε τη λειτουργία του συστήματος.



2.3.2 Fault Tree generation

Το Cecilia Ocas περιλαμβάνει **fault tree generator**. Αυτό το εργαλείο παράγει αποτελεσματικά μία Boolean φόρμουλα (ένα fault tree) το οποίο περιγράφει όλες τις επιπτώσεις των failure events του μοντέλου AltaRica.

Πλεονεκτήματα**:** Χάρη στην fault tree analysis ο μηχανικός ασφάλειας μπορεί να υπολογίσει τα ελάχιστα υποσύνολα μιας συνθήκης αποτυχίας και να επινοήσει ποιος είναι ο ελάχιστος αριθμός των failure events που οδηγούν σε αυτήν. Εάν τα ποσοστά εμφάνισης αποτυχίας συνδέονται με τα failure events του μοντέλου AltaRica, μπορεί να παρασταθεί επιπλέον και ποιοτική ανάλυση (probabilistic computations).

Περιορισμοί: Ο αλγόριθμος έχει ισχυρούς περιορισμούς όσον αφορά στη μορφή του μοντέλου AltaRica που θα ληφθεί υπόψη: η διάταξη εμφάνισης των γεγονότων στο μοντέλο δε θα πρέπει να έχει ως συνέπεια διαφορά στην κατάσταση του συστήματος. Έτσι δε μπορούμε να εφαρμόσουμε πάντα αυτό το εργαλείο στα μοντέλα. Προκειμένου να ξεπεράσουμε αυτόν τον περιορισμό, το Cecilia Ocas περιλαμβάνει επίσης ένα εργαλείο **sequence generator** το οποίο εξερευνεί το χώρο καταστάσεων του μοντέλου με σκοπό να βρει περιορισμένου μήκους ακολουθίες οι οποίες οδηγούν σε συνθήκες μιας δεδομένης αποτυχίας.

Η γλώσσα μοντελοποίησης AltaRica

2.4 Τυπικές βάσεις της AltaRica

Το ερευνητικό έργο AltaRica γεννήθηκε το 1996 από την επιθυμία των βιομηχάνων και των ερευνητών να δημιουργήσουν γέφυρες ανάμεσα στην ασφάλεια λειτουργίας και τις τυπικές μεθόδους, τις ποιοτικές αναλύσεις των δυσλειτουργιών και τις ποσοτικές αναλύσεις των λειτουργικών συστατικών, τα διάφορα εργαλεία και μεθόδους που βοηθούν στη μοντελοποίηση. Σκοπός της είναι η μοντελοποίηση κρίσιμων συστημάτων με ασφαλή λειτουργία.

Οι συνεργάτες: Οι κοινότητες IXI και ELF Aquitaine Production από τη μια πλευρά, τα εργαστήρια LaBRI (Laboratoire Bordelais de Recherche en Informatique) και LADS από την άλλη έχουν, κατά πρώτη φάση προσδιορίσει και επικυρώσει μέσω πειραμάτων δοκιμαστικές υποθέσεις της γλώσσας AltaRica.

Η δεύτερη φάση επέτρεψε τον προσδιορισμό της σημειολογίας της γλώσσας, να αναπτύξει πρωτότυπα των απαραίτητων εργαλείων στη προσομοίωση ενός μοντέλου AltaRica , να αναπτύξει πρωτότυπα των μεταγλωττιστών γύρω από τα εργαλεία Aralia και MEC και να περιγράψει ένα πρώτο εγχειρίδιο μεθοδολογίας.

Έκτοτε, το έργο ALTARICA οδήγησε στη γέννηση περισσότερων έργων.

- Η Dassault Aviation ανέπτυξε το εργαστήρι Cecilia OCAS στο οποίο βέβαιες αντιλήψεις τις γλώσσας ακατάλληλες για αυτούς και αλγοριθμικά ασύμφορες απαγορεύτηκαν, αλλά κάποια *patterns* μεταξύ των πιο χρησιμοποιούμενων αναπτύχθηκαν σε *macros*.
- Το LaBRI μαζί με άλλα εργαστήρια ερευνών συνεχίζει να μελετά και να επεκτείνει το μοντέλο. (http://altarica.labri.fr/)

Παράλληλα, αρκετοί πειραματισμοί όσον αφορά στη χρησιμοποίησή του διεκπεραιώθηκαν από διάφορους συνεργάτες του έργου.

Η εταιρία WideWave S.A. είναι άλλη μία νέα επιχείρηση. Ένα από τα έργα της είναι η πραγματοποίηση του ακόλουθου διπλώματος από έναν από τους δημιουργούς

της: Ένα σύστημα λειτουργίας από απόσταση για την απενεργοποίηση και ενεργοποίηση των εκπεμπόμενων σημάτων από ένα κινητό τηλέφωνο [14].

Η σημειολογία τής γλώσσας βασίζεται στα αυτόματα με περιορισμούς, μας ενδιαφέρουν η μοντελοποίηση συστημάτων και η ανάλυση της ασφάλειας διότι αυτά επιτρέπουν μια συγχρόνως λειτουργική και δυσλειτουργική μοντελοποίηση. Αυτή η δυσλειτουργική θεώρηση επιτρέπεται χάρη στην εισαγωγή γεγονότων (events) που μοντελοποιούν την αποτυχία των συστατικών. Η AltaRica επιτρέπει την ιεραρχική περιγραφή συστημάτων τα οποία απαρτίζονται από συστατικά τα οποία μπορούν να αλληλεπιδράσουν : είτε μέσω συγχρονισμού των ενεργειών των συστατικών χάρη στις εμφανίσεις των γεγονότων, είτε μέσω του συντονισμού των ροών των συστατικών χάρη στις εμφανίσεις ροών δεδομένων.

Η ικανότητά της να δημιουργεί μοντέλα σύνθεσης και ιεραρχικά, της επιτρέπει να μοντελοποιεί σύνθετα συστήματα. Διάφοροι μεταγλωττιστές γύρω από τα κλασικά εργαλεία αναλαμβάνουν να αναλύσουν είτε ποιοτικά είτε ποσοτικά το σύστημα.

Τέλος, η AltaRica είναι μία γλώσσα συγχρόνως κατηγορηματική και γραφική χάρη στο OCAS (Outil de Conception et d’Analyse Système) που αναπτύχθηκε από το Dassault το οποίο επιτρέπει τον προγραμματισμό καθώς και την προσομοίωση μοντέλων.

Πέρα από το Ocas που μόλις αναφέραμε αξίζει να αναφέρουμε ονομαστικά τουλάχιστον και κάποια άλλα εργαλεία της γλώσσας. Τα χωρίζουμε σε δύο κατηγορίες:

<u>Workbenches:</u>

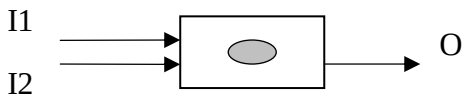
- Δυναμικές διασυνδέσεις χρήστη για το σχεδιασμό μοντέλων
- Γραφικοί προσομοιωτές
- OCAS (Dassault Aviation), SimFia (EADS-APSYS), Saraa (Airbus)

<u>Assessment tools:</u>

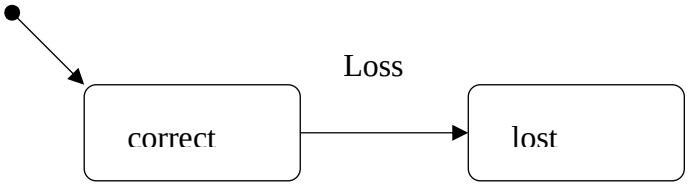
- Μεταγλωττιστές για fault trees
- Μεταγλωττιστές για γράφους Marcov.
- Στοχαστικούς προσομοιωτές.
- Γεννήτριες ακολουθιών.
- Μεταγλωττιστές για μεθοδικές γλώσσες (Lustre, SMV).
- Ελεγκτές μοντέλων
- AltaTools, Mec V (LaBRI), Combava (ARBoost Technologies)

2.5 Ιδέα

Κάθε συστατικό στην AltaRica, ονομαζόμενο ως κόμβος (node), συντίθεται από δηλώσεις μεταβλητών και γεγονότων, καθώς και τον ορισμό μεταβάσεων (transitions) και ισχυρισμών (assertions). Μπορούμε επίσης να χρησιμοποιήσουμε έναν κόμβο στο εσωτερικό ενός άλλου κόμβου (sub) και να τους κάνουμε να επικοινωνούν. Απεικονίζουμε τις αρχές αυτές με το ακόλουθο παράδειγμα. Το συστατικό Function2 έχει δύο εισόδους I1 και I2 και μία έξοδο O. Όσο βρίσκεται στη σωστή κατάσταση, ο κόμβος παράγει μία σωστή (correct) έξοδο όταν και οι δύο είσοδοι είναι σωστές, δεν παράγει κάποια τιμή αν και οι δύο είναι είσοδοι χαθούν αλλιώς παράγει μία λανθασμένη τιμή ως έξοδο. Αν ο κόμβος Function2 βρίσκεται σε σωστή κατάσταση λειτουργίας, μπορεί να συμβεί μία βλάβη και να αλλάξει την κατάσταση του κόμβου σε λανθασμένη (lost). Σε αυτήν την λανθασμένη κατάσταση, ο Function2 δεν παράγει κάποια τιμή [2].



Εικόνα 1: Γραφική αναπαράσταση του κόμβου Function2



Εικόνα 2: Κατάσταση αποτυχίας του αυτόματου για τον κόμβο Function2

Το μοντέλο του κόμβου Function2 παρουσιάζεται στον ακόλουθο πίνακα..

Node		
Function2		
State	**Type**	**initial value**
Status	FailureType	correct
Flow	**Type**	**direction**
I1	FailureType	in
I2	FailureType	in
O	FailureType	out
Event	**failure rate**	
Loss	exp 1.0e-4	
Transition	**guard**	**new affectations**
		Status
Loss	Status != lost	lost
Assertion	**case**	**value**
O	Status=correct	correct
	I1=correct	
	I2=correct	
	else	lost

Στο τμήμα state δηλώνουμε μεταβλητές κατάστασης (state variables) δίνοντας

- το όνομα της μεταβλητής κατάστασης : Status,
- το πεδίο τιμών της μεταβλητής Status : είναι στο πεδίο FailureType που όρισε ο χρήστης,
- την αρχική κατάσταση : η μεταβλητή Status αρχικά βρίσκεται στην κατάσταση correct.

Τα πεδία τιμών που χρησιμοποιούμε είναι : Boolean, enumeration και record type που διαμορφώνονται από Boolean και enumeration. Για παράδειγμα, το πεδίο Failure-Type είναι τύπου enumeration {correct, lost}. Ένα ενδιαφέρον σημείο στην AltaRica είναι ότι οι τιμές των μεταβλητών είναι πάντα διακριτές.

Στο τμήμα flow δηλώνουμε μεταβλητές οι οποίες χρησιμοποιούνται στο μοντέλο για τις ανταλλαγές δεδομένων μεταξύ των συστατικών που αλληλεπιδρούν δίνοντας :

- το όνομα και τον τύπο της μεταβλητής ροής,
- την κατεύθυνση της μεταβλητής ροής : in για είσοδο, out για έξοδο, local για μεταβλητές που χρησιμοποιούνται ως συντομογραφίες.

Στο τμήμα event δηλώνουμε γεγονότα και λέμε εάν πυροδοτούνται εσωτερικά από το σύστημα ή εξωτερικά από το περιβάλλον τους. Μπορούμε να συνδυάσουμε έναν κανόνα πιθανότητας εμφάνισης με ένα γεγονός.

Το τμήμα transition περιέχει το σύνολο μεταβάσεων που περιγράφουν τα γεγονότα. Μία μετάβαση συντίθεται από:

- το όνομα του γεγονότος που πυροδοτεί τη μετάβαση, για παράδειγμα Loss,
- μία συνθήκη (guard) η οποία είναι λογική έκφραση που βασίζεται σε μεταβλητές κατάστασης και μεταβλητές input/local ορίζοντας τις συνθήκες που θα πυροδοτηθεί η μετάβαση,
- και το σύνολο με νέες επιδράσεις στις μεταβλητές κατάστασης.

Στον προηγούμενο πίνακα, μία μετάβαση συνδέεται με το γεγονός Loss το οποίο πυροδοτείται μόνον όταν το συστατικό Function2 δεν είναι σε κατάσταση lost (π.χ. Status είναι διάφορο του lost). Η νέα τιμή της Status είναι lost. Η τιμή μιας μεταβλητής κατάστασης μπορεί να τροποποιηθεί μόνο μέσω μεταβάσεων.

Το τμήμα assertion περιέχει το σύνολο των κανόνων υπολογισμού των τιμών των output/local μεταβλητών ροής. Ο κανόνας υπολογισμού της τιμής μιας μεταβλητής εξόδου μπορεί να βασίζεται σε μεταβλητές κατάστασης εισόδου και local. Ο προηγούμενος πίνακας προσδιορίζει πώς παράγεται η μεταβλητή εξόδου O. Μία correct έξοδος παράγεται (O=correct) εάν η Status και οι δύο μεταβλητές εισόδου είναι όλες ίσες με την τιμή correct αλλιώς δε στέλνεται καμία έξοδος (O=lost).

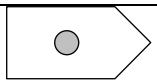
Στη συνέχεια παρουσιάζεται ο κώδικας για τον κόμβο Function2.

```
node COMPUTATION_Function2
    flow
        O : COMPUTATION_FailureType : out;
```

```
        I1 : COMPUTATION_FailureType : in;
        I2 : COMPUTATION_FailureType : in;
    state
        Status : COMPUTATION_FailureType;
    event
        Loss;
    trans
        (Status != lost) |- Loss -> Status := lost;
    assert
        O = case {
            (((Status = correct) and (I1 = correct)) and (I2 = correct)) : correct, else lost
        };
    init
        Status := correct;
    extern
        law (<event Loss>) = “exp 1e-4”;
edon
```

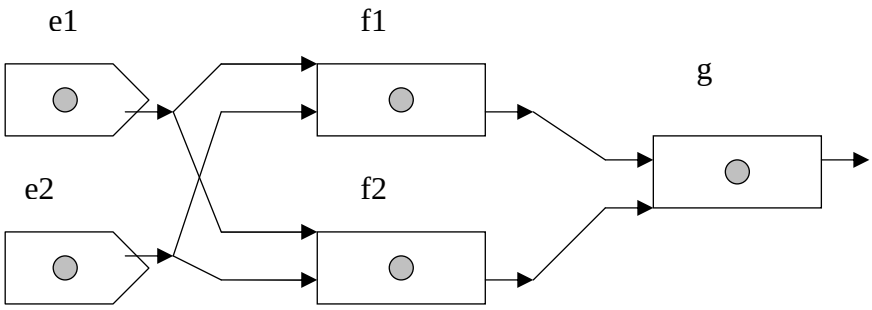
Μοντέλο σε AltaRica

Τα μοντέλα της AltaRica κατασκευάζονται από διασυνδεδεμένα στιγμιότυπα κόμβων της γλώσσας. Για παράδειγμα, στην ακόλουθη εικόνα, τρία στιγμιότυπα του κόμβου Function2 με τις ονομασίες f1, f2 και g συνδέονται. Επιπλέον συνδέονται με δύο στιγμιότυπα (e1, e2) του κόμβου Source. Ο κόμβος Source παρουσιάζεται στη συνέχεια.

Node		
Source		
		
Node	Type	Initial Value

State	Type	Initial Value
State	Type	Initial Value
Status	FailureType	correct
Flow	Type	Direction
O	FailureType	Out
Event	Failure Rate	
loss	exp 10e-4	
error	exp 10e-5	
Transition	Guard	New affectations
		Status
loss	Status != lost	lost
		Status
error	Status != erroneous	Erroneous
Assertion	Case	Value
O	true	Status

Πίνακας 1: Περιγραφή του κόμβου Source



Εικόνα 3: Διασύνδεση κόμβων της AltaRica

Τέτοιου είδους διασύνδεση κόμβων μπορεί να μετασχηματιστεί σε έναν επίπεδο κόμβο AltaRica. Η κλασική σημειογραφία με τελείες χρησιμοποιείται για να δηλώσει

τις μεταβλητές και τα γεγονότα ενός στιγμιοτύπου : g.Status είναι η Status μεταβλητή του κόμβου g και f1.loss είναι το γεγονός loss του στιγμιοτύπου f1. Ένας σύνδεσμος μεταξύ δύο μεταβλητών σημαίνει ότι αυτές οι μεταβλητές θα πρέπει πάντοτε να έχουν την ίδια τιμή. Για να το μοντελοποιήσουμε αυτό, οι συνδεδεμένες μεταβλητές εισόδου – εξόδου μετασχηματίζονται σε local μεταβλητές, και εξισώσεις συνδέσεων (όπως, g.In1 = f1.O) προστίθενται στο τμήμα assertion του κόμβου.

Node Mode11		
state	type	initial value
e1.Status	FailureType	correct
e2.Status	FailureType	correct
f1.Status	FailureType	correct
f2.Status	FailureType	correct
g.Status	FailureType	correct
flow	type	direction
g.O	FailureType	out
event	failure rate	
e1.loss	exp 1.0e-4	
e2.loss	exp 1.0e-4	
f1.loss	exp 1.0e-4	
f2.loss	exp 1.0e-4	
g.loss	exp 1.0e-4	
transition	guard	new affectations
		e1.Status
e1.loss	e1.Status != lost	lost
		e2.Status
e2.loss	e2.Status != lost	lost

		f1.Status
f1.loss	f1.Status != lost	lost
		f2.Status
f2.loss	f2.Status != lost	lost
		g.Status
g.loss	g.Status != lost	lost
assertion	case	value
g.O	g.Status=correct and f1.Status=correct and f2.status=correct and e1.Status=correct and e2.Status	correct
	else	lost

Πίνακας 2: Επίπεδη περιγραφή του μοντέλου AltaRica

Αν και η διασύνδεση μεταβλητών ροής είναι ο πιο εύχρηστος τρόπος για να περιγράψουμε διασυνδέσεις σε έναν κόμβο AltaRica, είναι επίσης δυνατό να συνδέσουμε ομάδες γεγονότων από διαφορετικούς κόμβους. Υπάρχουν δύο τρόποι συνδέσεων γεγονότων:

- Strong Synchronisation: το συγχρονισμένο γεγονός μπορεί να πυροδοτηθεί αν ολόκληρη η ομάδα γεγονότων μπορεί να πυροδοτηθεί (όλες οι συνθήκες των μεταβάσεων που σχετίζονται με τα γεγονότα γίνουν αληθείς). Όταν το συγχρονισμένο γεγονός πυροδοτηθεί ολόκληρη η ομάδα των γεγονότων πυροδοτείται και όλες οι μεταβλητές κατάστασης τροποποιούνται σύμφωνα με τις μεταβάσεις των ομαδοποιημένων γεγονότων.
- Broadcast: το broadcast event μπορεί να πυροδοτηθεί εάν τουλάχιστον ένα γεγονός από το σύνολο των ομαδοποιημένων γεγονότων μπορεί να πυροδοτηθεί. Όταν το broadcast event πυροδοτείται όλα τα ομαδοποιημένα γεγονότα τα οποία

μπορούν να πυροδοτηθούν πυροδοτούνται και μεταβάλλονται οι μεταβλητές κατάστασης.

Και στις δύο περιπτώσεις μπορούμε να αποφασίσουμε αν ομαδοποιημένα γεγονότα μπορούν να ¨κρυφτούν¨ ή όχι. Ένα γεγονός το οποίο ανήκει σε μια ομάδα strong synchronization ή broadcast είναι ¨ κρυφό¨ εάν δε μπορεί να πυροδοτηθεί ατομικά, έτσι απομακρύνεται από τη λίστα των γεγονότων του επίπεδου κόμβου. Το εργαλείο OCAS υποστηρίζει μόνο Strong Synchronisation με όλα τα ομαδοποιημένα γεγονότα ¨κρυφά¨, Broadcast με όλα τα ομαδοποιημένα γεγονότα ¨κρυφά¨ και Broadcast με όλα τα ομαδοποιημένα γεγονότα ¨όχι κρυφά¨ (αυτό ονομάζεται CCF –Common Cause Failure).

Ο ακόλουθος πίνακας παρουσιάζει τον κόμβο που σχετίζεται με το προηγούμενο μοντέλο όπου το γεγονός ss_e_loss είναι strong synchronization του e1.loss και e2.loss, και bc_f_loss είναι broadcast των γεγονότων f1.loss και f2.loss. Όλα τα ομαδοποιημένα γεγονότα είναι ¨κρυφά¨.

event	failure rate		
g.loss	exp 1.0e-4		
ss_e_loss	exp 1.0e-8		
bc_f_loss	exp 1.0e-8		
transition	guard	new affectations	
		e1.Status	e2.Status
ss_e_loss	e1.Status != lost and e2.Status != lost	lost	lost
		f1.Status	f2.Status
bc_f_loss	f1.Status != lost or f2.Status != lost	if (f1.Status != lost) then lost else f1.Status	if (f2.Status != lost) then lost else f2.Status
		g.Status	
g.loss	g.Status != lost	lost	

Κάθε κόμβος μπορεί να περιγραφεί σαν ένα αυτόματο με περιορισμούς. Κάθε κόμβος του αυτόματου είναι μία σύνθεση τμημάτων, όπως μία διαδικασία η οποία αποδίδει μία τιμή σε κάθε μεταβλητή κατάστασης και ροής. Στην πραγματικότητα, η συμπεριφορά του συστατικού περιγράφεται από τους περιορισμούς στις αλλαγές καταστάσεων, επονομαζόμενους transitions, καθώς και από τους καθολικούς περιορισμούς στις μεταβλητές, επονομαζόμενους assertions. Η αρχική διαμόρφωση προσδιορίζεται από την αρχική τιμή των μεταβλητών κατάστασης των συστατικών. Έπειτα, μέσω των δηλώσεων στο τμήμα assertion αποδίδονται τιμές στις εξόδους. Αυτές οι τιμές εκπέμπονται στα συστατικά που συνδέονται και έτσι οι τιμές αυτές αποδίδονται στις εισόδους των συστατικών.

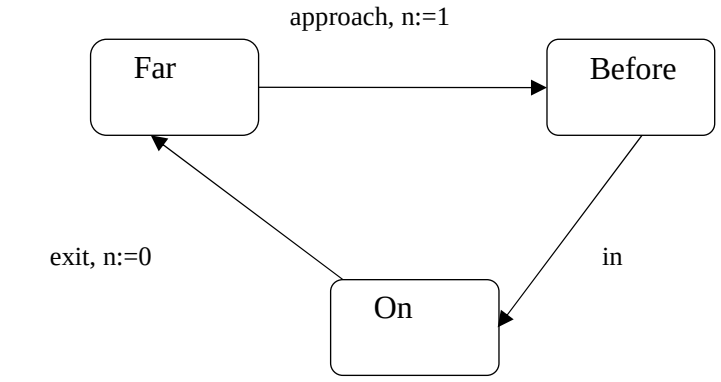
Ένα σενάριο του μοντέλου είναι ένα μονοπάτι στο αυτόματο το οποίο αρχίζει από την αρχική διαμόρφωση και κινείται από σχηματισμό σε σχηματισμό με την επιλογή γεγονότων.

Παράδειγμα**:** Στόχος είναι να κρατήσουμε τη θύρα κλειστή όσο ένα τρένο είναι στο κρίσιμο τμήμα.

Προσδιορισμός του τρένου στην AltaRica:

```
node TRAIN
    flow N : [0,1];  //μεταβλητή εξόδου
    event approach, in, exit;
    state etat : [0, 2]; n : [0,1];
    trans
        etat = 0 |- approach -> etat := 1, n := 1;
        etat = 1 |- in -> etat := 2;
        etat = 2 |- exit ->etat := 0, n := 0;
    init
        etat :=0, n :=0;
    assert
        N = n;
edon
```

Προσδιορισμός του τρένου ως αυτόματο:



Far ≡ etat = 0

Before ≡ etat = 1

On ≡ etat = 2

Ορίσαμε ένα συστατικό train ώστε να μοντελοποιήσουμε τη συμπεριφορά ενός τρένου με δύο ισοδύναμους τρόπους έτσι ώστε να διευκολύνουμε την κατανόηση. Ένα τρένο είτε είναι Far από το κρίσιμο τμήμα είτε Before είτε On που σημαίνει ότι είναι ή κοντά ή μέσα στην κρίσιμη ζώνη. Στην AltaRica, η μεταβλητή etat που κυμαίνεται στο διάστημα τιμών [0, 2] αναπαριστά τις τοποθετήσεις Far, Before, On του τρένου. Τα γεγονότα του συστατικού TRAIN είναι τα approach, in και exit. Επίσης, χρησιμοποιούμε μία μεταβλητή κατάστασης n για να δηλώσουμε ότι το τρένο είναι στις καταστάσεις {Before, On}. Αρχικά, οι τιμές των μεταβλητών του συστατικού διαμορφώνονται ως etat = 0, n = 0, N = 0, γράφοντας (0,0,0) για συντομία. Όταν συμβεί μια μετάβαση οι τιμές των μεταβλητών κατάστασης αλλάζουν σύμφωνα με την τιμή των μεταβλητών ροής ώστε να ικανοποιήσουν τις δηλώσεις στο τμήμα assertion. Για παράδειγμα, όταν συμβεί το γεγονός approach στην κατάσταση (0, 0 , 0) έχουμε τη διαμόρφωση σε (1, 1, 1).

Διασυνδέσεις

Οι μεταβλητές κατάστασης του συστατικού δεν είναι ορατές από το εξωτερικό του συστατικού. Για να επιτραπεί η διανομή πληροφορίας και ο συγχρονισμός με μεταβλητές άλλων συστατικών χρησιμοποιούμε τις μεταβλητές ροής. Οι μεταβλητές ροής

μπορούν να διαβαστούν και από άλλους κόμβους. Το μέρος του συστατικού που είναι ορατό από άλλα συστατικά ονομάζεται *διασύνδεση*.

Η μεταβλητή ροής N είναι στη διασύνδεση του κόμβου TRAIN. Αυτό σημαίνει ότι οι άλλοι κόμβοι μπορούν να τη διαβάσουν και να χρησιμοποιήσουν την τιμή του N. Η τιμή του N είναι κάθε στιγμή ίση με αυτήν του n και ο λόγος ύπαρξης του N είναι για να κάνει την τιμή του n διαθέσιμη στο εξωτερικό του συστατικού.

Θεωρούμε έναν άλλο κόμβο για τον ελεγκτή (controller). Ο ορισμός του στην AltaRica υπάρχει παρακάτω. Μια μετάβαση τύπου etat = 1 |- approach -> ; σημαίνει ότι το γεγονός approach δεν επιφέρει καμία αλλαγή στις τιμές των μεταβλητών κατάστασης. Στο συστατικό CONTROLLER ο σκοπός της μεταβλητής ροής N (αναφέρεται ως CONTROLLER .N) είναι να μετράει το συνολικό αριθμό τρένων στην περιοχή {Before, On}. Ανάλογα με την τιμή της CONTROLLER .N ο ελεγκτής θα ανεβάσει τη θύρα με ένα γεγονός exit (αν η τιμή ισούται με 1) ή θα αφήσει τη θύρα κλειστή αν CONTROLLER .N >1.

Εκτός από τα γεγονότα που υπάρχουν στο τμήμα event του συστατικού, θεωρούμε ένα ειδικό διακριτό γεγονός ϵ για λόγους συγχρονισμού. Το γεγονός αυτό δεν αλλάζει την τιμή των μεταβλητών κατάστασης.

Ο κόμβος GATE που υλοποιείται στη συνέχεια, λαμβάνει εντολές από τον ελεγκτή (γεγονότα Go_up και Go_down) και ανεβαίνει ή κατεβαίνει (γεγονότα up, down).

```
node CONTROLLER
    flow N : [0,p];
    event approach, exit, Go_up, Go_down;
    state etat : [0,2];
    trans
        etat = 0 |- approach -> etat := 1;
        etat = 0 & N>1 |- exit -> ;
        etat = 0 & N = 1 |- exit -> etat := 2;
        etat = 1 |- approach -> ;
        etat = 1 |- exit -> ;
        etat = 1 |- Go_down ->etat := 0;
```

```
        etat = 2 |- Go_up ->etat := 0;
        etat = 2 |- approach ->etat := 1;
    init etat := 0, z := 0;
edon

node GATE
    event Go_down, Go_up, down, up;
    state etat : [0, 3];
    trans
        etat = 0 |- Go_up ->;
        etat = 0 |- Go_down -> etat := 1;
        etat = 1 |- Go_down -> ;
        etat = 1 |- down -> etat := 2 ;
        etat = 1 |- Go_up -> etat := 3;
        etat = 2 |- Go_down -> ;
        etat = 2 |- Go_up ->etat := 3;
        etat = 3 |- Go_up -> ;
        etat = 3 |- Go_down -> etat := 1;
        etat = 3 |- up -> etat := 0;
    init etat := 0;
edon

node MAIN
sub
    t1, t2 : TRAIN;
    g : GATE;
    c : CONTROLLER;
sync
    <t1.approach, t2.approach, c.approach>;
```

```
    <t1.approach, c.approach>;
    <t2.approach, c.approach>;
    <t1.exit, t2.exit, c.exit>;
    <t1.exit, c.exit>;
    < t2.exit, c.exit>;
    <g.Go_down, c.Go_down>;
    <g.Go_up, c.Go_up>;
assert c.N = t1.N + t2.N;
edon
```

Παράδειγμα**:** Θέλουμε να μοντελοποιήσουμε μία ηλεκτρική πηγή. Η δυαδική του κατάσταση (λειτουργία, δυσλειτουργία) θα αντιπροσωπευθεί από μία μεταβλητή κατάστασης που τη σημειώνουμε με s. Μία έξοδος του συστήματος που τη σημειώνουμε ο, παρέχει μία τιμή λογική στη συνάρτηση της κατάστασης της πηγής. Μία αποτυχία fail μπορεί να έχει ως συνέπεια την αποτυχία του συστήματος. Σε αυτήν την περίπτωση το σύστημα δεν παράγει έξοδο. Η συμπεριφορά του συστήματος εξαναγκάζεται με δύο τρόπους. Μπορεί να αλλάξει την εσωτερική κατάσταση με τρόπο διακριτό από τις μεταβάσεις που προκαλούνται από την τρέχουσα κατάσταση, η τιμή των μεταβλητών ροής και των γεγονότων (event) κλασικών ή ειδικών (τα οποία σημειώνονται με ε) τα οποία δεν προσδιορίζονται στο μοντέλο αυτό καθ’ αυτό αλλά προσδιορίζονται σε σημειολογικό επίπεδο και τα οποία επιτρέπουν τη διαφύλαξη ενός κόμβου από τον αποκλεισμό. Όσον αφορά στα assertions, είναι σχέσεις εξάρτησης ανάμεσα στην κατάσταση του συστατικού και στην τιμή των μεταβλητών ροής. Εδώ είναι ο κώδικας που αντιστοιχεί σε αυτό το συστατικό.

```
 node source
state s:bool;
flow o:out:bool;
event fail;
trans s |- fail -> s:=false;
assert o=s;
```

init s:=true;

edon

Αυτές οι θεωρήσεις ορίζονται τυπικά από τα αυτόματα με περιορισμούς.

Ορισμός. Ένα αυτόματα με περιορισμούς A είναι μια εννιάδα < D, S, F^{in}, F^{out}, dom, Σ, δ, σ, I> με:

D: πεπερασμένο σύνολο σταθερών, ονομαζόμενο domain

Συμβολίζουμε V ένα πεπερασμένο σύνολο μεταβλητών που χωρίζονται σε τρεις κατηγορίες S, F^{in}, F^{out} . Αυτές οι μεταβλητές είναι υποσύνολα του V ανά δύο χωρισμένες που ονομάζονται αντίστοιχα μεταβλητές κατάστασης, ροής εισόδου και ροής εξόδου.

Οι μεταβλητές κατάστασης δείχνουν, όπως δηλώνει και το όνομά τους, την εσωτερική κατάσταση του συστατικού. Οι μεταβλητές ροής αντιστοιχούν σε βοηθητικές τιμές , οι οποίες δεν αποτελούν μέρος της κατάστασης του συστατικού, και οι οποίες επιτρέπουν στο συστατικό να επικοινωνεί με το περιβάλλον του.

Όπως δηλώθηκε και προηγουμένως, αυτοί οι δύο τύποι των μεταβλητών επεμβαίνουν για να εξαναγκάσουν τη συμπεριφορά του συστατικού σε transitions και assertions.

dom: V -> 2^D έτσι ώστε $\forall v \in V$, $dom(v) \neq \emptyset$ συνδέει σε μία μεταβλητή το domain της

Σ είναι το πεπερασμένο σύνολο γεγονότων

δ είναι μία μερική λειτουργία που ονομάζεται transition : $dom(S) \times dom(F^{in}) \times \Sigma$ -> dom(S)

σ είναι μία ολική λειτουργία που ονομάζεται assertion : $dom(S) \times dom(F^{in})$-> $dom(F^{out})$

$I \subset \sigma$ είναι μία μερική λειτουργία η οποία περιγράφει τις αρχικές συνθήκες

Οι καταστάσεις και οι μεταβάσεις ορίζονται από τα δ και σ. Σημειώνουμε ότι υπάρχει ένα πεπερασμένο σύνολο αφού οριστούν τα V και D.

Προτεραιότητες

Στην AltaRica μπορούμε να περιορίσουμε τις συμπεριφορές ενός συστήματος δίνοντας προτεραιότητες σε κάποιες μεταβάσεις όταν περισσότερες από μία είναι δυνατές. Τυπικά, μία σχέση προτεραιότητας < είναι μία ακριβής διάταξη στα γεγονότα. Μια μετάβαση e μπορεί να πυροδοτηθεί από μία διαμόρφωση (s, f) αν είναι η ανώτατη (maximal), για παράδειγμα καμία άλλη μετάβαση e΄ τέτοια ώστε e < e΄ δεν πυροδοτείται στη (s, f).

Ορισμός συστατικού:

Ένα συστατικό είναι μια εξάδα C = ‹Vs, Vf, E, A, M, < › με :

1. *Vs, Vf* είναι πεπερασμένα σύνολα μεταβλητών κατάστασης, μεταβλητών ροής.
2. $E = E_+ \cup \{\varepsilon\}$ είναι ένα πεπερασμένο σύνολο γεγονότων και ε είναι η κενή ενέργεια.
3. $A \in \mathbb{F}$ είναι μία assertion τέτοια ώστε $free(A) \subseteq V_C$.
4. $M \subseteq \mathbb{F} \times E \times \mathbb{E}(V_C)^{VC}$ είναι μία *macro-transition σχέση τέτοια* ώστε $(tt, \varepsilon, Id) \in M$ και κάθε $(g, e, a) \in M$ ικανοποιεί :
    a) $g \in \mathbb{F}$ είναι μία *guard* τέτοια ώστε $free(g) \subseteq V_C$,
    b) $e \in E_+$ είναι το γεγονός της μετάβασης,
    c) $a : Vs \to \mathbb{E}(V_C)$ είναι μία εκχώρηση (assignment) για τις μεταβλητές του *Vs*.
5. < είναι μια σχέση προτεραιότητας.

3 Ανάπτυξη του μοντέλου

3.1 Κόμβοι του δικτύου και Distributed Signal Boxes

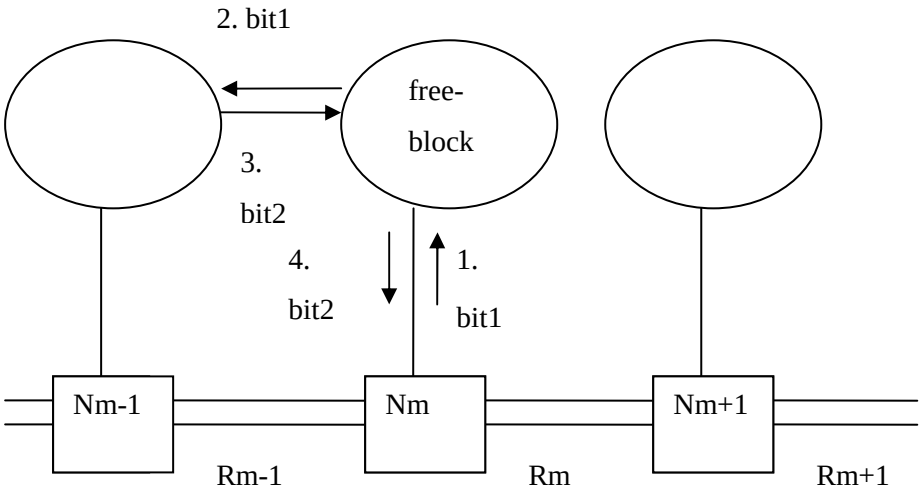
Τα DSBs συνδέονται με τα διασυνδεμένα σημεία του δικτύου - στο εξής τα ονομάζουμε κόμβους (nodes), τα οποία δε μπορούν να επικοινωνούν μεταξύ τους, αλλά μπορούν να επικοινωνούν με τα αντίστοιχα DSBs τους. Κάθε DSB επικοινωνεί με τα γειτονικά του DSBs.

3.1.1 Αλγόριθμος για τον έλεγχο μέσω DSBs

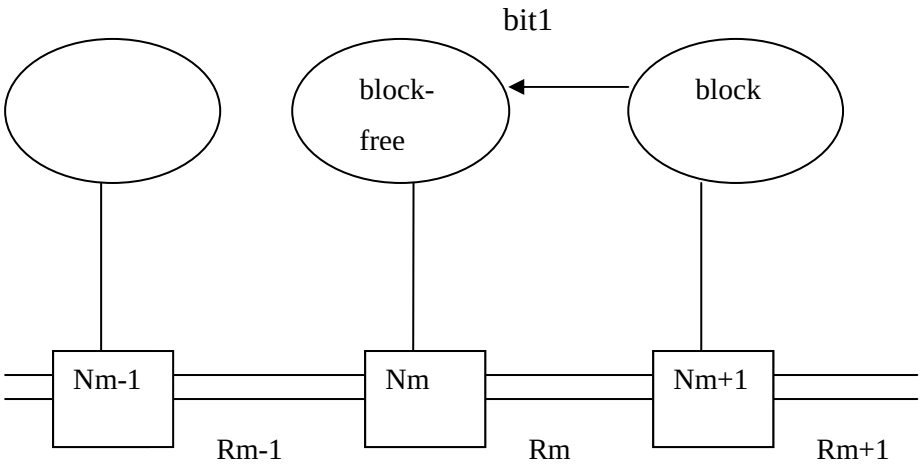
Υποθέτουμε ότι μία οντότητα Ε βρίσκεται στο κανάλι Rm-1 και θέλει να πάει στο κανάλι Rm, το οποίο ελέγχεται από τον κόμβο Nm. Ο κόμβος Nm τότε στέλνει ένα μήνυμα bit1 στο αντίστοιχο DSB του. Διακρίνουμε δύο περιπτώσεις:

- Το κανάλι Rm είναι διαθέσιμο. Στην περίπτωση αυτή το DSB του κόμβου Nm στέλνει ένα μήνυμα bit1 στο DSB του σταθμού Nm-1 και το DSB του κόμβου αυτού του απαντάει με ένα μήνυμα bit2 εφόσον ο κόμβος Nm-1 δεν έχει κανένα τρένο. Το bit2 λαμβάνεται από το DSB του Nm και στη συνέχεια στέλνεται στον κόμβο Nm ο οποίος στη συνέχεια παρέχει πρόσβαση στο κανάλι Rm. Το DSB του κόμβου Nm μπλοκάρεται περιμένοντας για το μήνυμα bit1 από το DSB του κόμβου Nm+1.
- Το κανάλι Rm είναι απασχολημένο από μία άλλη οντότητα. Στην περίπτωση αυτή δεν υπάρχει μήνυμα bit2 στο DSB του κόμβου Nm όπως επίσης και η παρούσα κατάσταση του σταθμού δηλώνει ότι ήδη υπάρχει ένα τρένο στο κανάλι κι έτσι μπλοκάρεται η αιτούσα οντότητα από την πρόσβαση στο κανάλι Rm. Το DSB του κόμβου Nm περιμένει επίσης για το μήνυμα bit1. Το αναμενόμενο μήνυμα θα ληφθεί όταν το DSB του κόμβου Nm+1 που ελέγχει το κανάλι Rm+1 στείλει bit1 δηλώνοντας έτσι ότι το κανάλι Rm+1 μπορεί να χρησιμοποιηθεί από την οντότητα που βρίσκεται στο κανάλι Rm. Επίσης, μόλις φύγει το τρένο που υπήρχε στο κανάλι η κατάσταση του σταθμού αλλάζει δηλώνοντας ότι ο σταθμός έχει ξεμπλοκαριστεί και το DSB του κόμβου Nm στέλνει bit2 στον αντίστοιχο κόμβο του και το μήνυμα αυτό ξεμπλοκάρει το αιτούμενο κανάλι Rm [5].

Ακολούθως παραθέτουμε ενδεικτικά δύο απεικονίσεις λειτουργίας των DSBs.



Διαγραμματική απεικόνιση για την πρώτη περίπτωση. Το DSB του κόμβου Nm αρχικά είναι ελεύθερο και στη συνέχεια μπλοκάρεται με την αίτηση του τρένου.



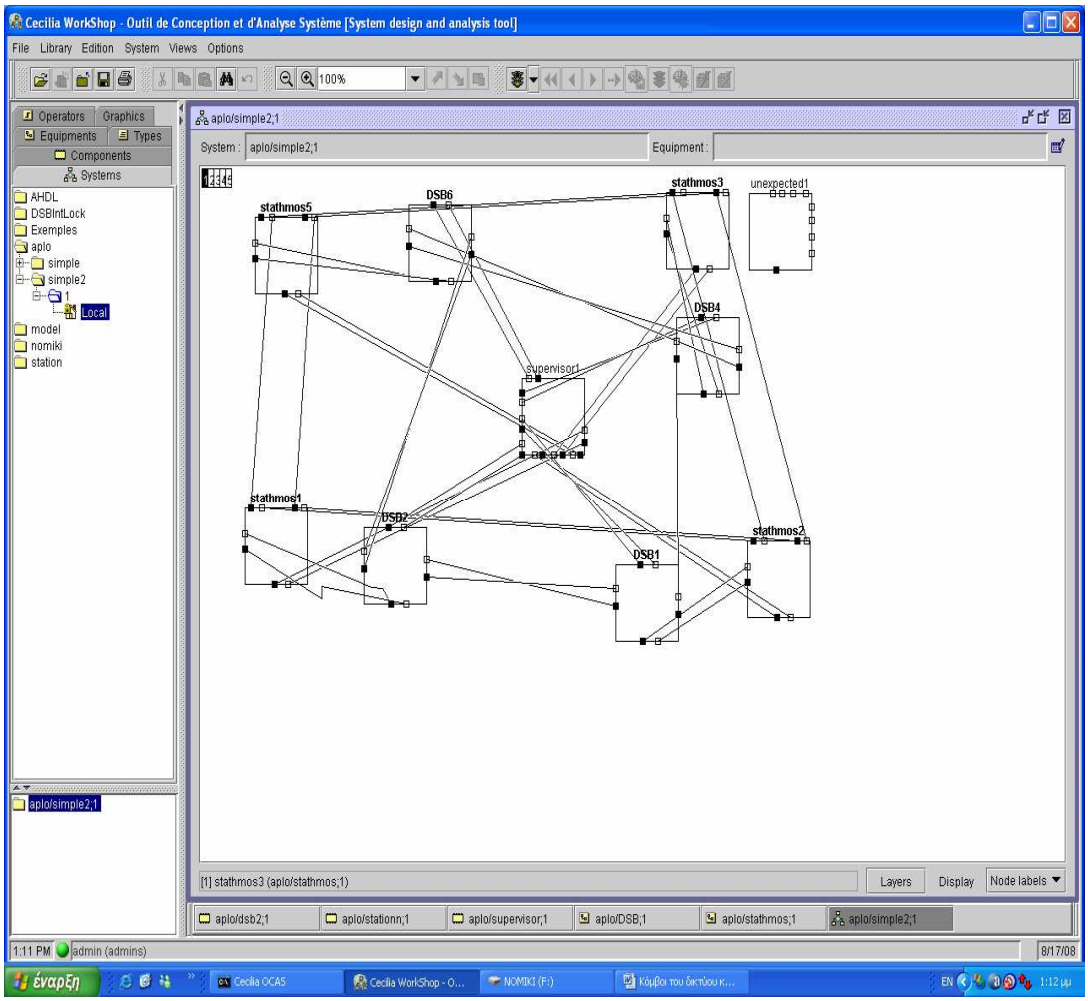
Διαγραμματική απεικόνιση για τη δεύτερη περίπτωση. Το DSB του κόμβου Nm ελευθερώνεται αμέσως μετά τη λήψη του μηνύματος bit1 από το DSB τού κόμβου Nm+1.

3.2 Δομή του μοντέλου και υλοποίηση

Για ένα σταθμό, έστω Χ, ένα τρένο που φτάνει σε αυτόν προκαλεί την εκπομπή ενός μηνύματος bit1 στο signal box του. Το signal box του Χ με τη σειρά του στέλνει το bit1 στο signal box του προηγούμενου σταθμού. Ο προηγούμενος σταθμός μόλις λάβει το μήνυμα bit1 και είναι ελεύθερος στέλνει ένα μήνυμα bit2 στον επόμενό του. Με την παραλαβή του bit2 από τον προηγούμενο σταθμό το τρένο εισέρχεται στο επόμενο κανάλι.. Πέρα από το μπλοκάρισμα των DSBs, ο κόμβος του κάθε σταθμού έχει ανά πάσα στιγμή μία κατάσταση η οποία δηλώνει εάν υπάρχει ή όχι τρένο σε αυτόν. Κάθε στιγμή σε κάθε κανάλι πρέπει να υπάρχει το πολύ ένα τρένο. Σε περίπτωση που τοποθετήσουμε ένα δεύτερο τρένο σε ένα σταθμό τότε δε συμβαίνουν οι ανταλλαγές μηνυμάτων που συνέβησαν με την εμφάνιση του πρώτου τρένου στο σταθμό. Αυτό συμβαίνει για να εμποδίσουμε την ταυτόχρονη απομάκρυνση και των δύο τρένων από το σταθμό. Επίσης, σε περίπτωση που υπάρχει ένα τρένο σε ένα σταθμό Χ, δεν είναι δυνατή από το σύστημα η απομάκρυνση κάποιου τρένου από τον προηγούμενο σταθμό με πορεία προς τον σταθμό Χ έως ότου αναχωρήσει το τρένο που βρίσκεται στον Χ.

Η τοπολογία του σιδηροδρομικού δικτύου που υλοποιήσαμε έχει τη μορφή τετραγώνου. Αποτελείται από τέσσερεις κόμβους – σταθμούς οι οποίοι συνδέονται ο καθένας με τον επόμενό του και από τέσσερα DSBs το καθένα από τα οποία συνδέεται με τον αντίστοιχο σταθμό του καθώς επίσης και με τα δύο γειτονικά του DSBs. Στην εικόνα 1 παρουσιάζεται η μοντελοποίηση του δικτύου χρησιμοποιώντας το εργαλείο Cecilia Ocas. Με τις λεπτές γραμμές απεικονίζονται όλες οι διασυνδέσεις μεταξύ των συστατικών ενώ με τα τετράγωνα πλαίσια οι οντότητες του συστήματος.

Εικόνα 1.


Cecilia WorkShop - Outil de Conception et d'Analyse Système [System design and analysis tool]
File Library Edition System Views Options
Operators Graphics
Equipments Types
Components
Systems
AHDL
DSBIntLock
Exemples
aplo
simple
simple2
1
Local
model
nomiki
station
aplo/simple2;1
System : aplo/simple2;1
Equipment :
stathmos5
DSB6
stathmos3
unexpected1
DSB4
supervisor1
stathmos1
DSB2
DSB1
stathmos2
[1] stathmos3 (aplo/stathmos;1)
Layers
Display
Node labels
aplo/dsb2;1
aplo/stationn;1
aplo/supervisor;1
aplo/DSB;1
aplo/stathmos;1
aplo/simple2;1
1:11 PM
admin (admins)
8/17/08
έναρξη
Cecilia OCAS
Cecilia WorkShop - O...
NOMIKI (F:)
Κόμβοι του δικτύου κ...
EN
1:12 μμ

Supervisor:

Στη μοντελοποίηση του δικτύου –η οποία παρουσιάζεται και στην εικόνα 1 χρησιμοποιήσαμε και μία άλλη οντότητα, αυτή του supervisor. Ας εξηγήσουμε ποιος είναι ο σκοπός τής οντότητας αυτής. Ο supervisor είναι υπεύθυνος για την επιδιόρθωση κάποιου συστατικού εάν αυτό υποστεί βλάβη καθώς και για την αλλαγή ρόλων (active, shadow) δύο συστατικών. Δηλαδή, σε περίπτωση που συμβεί ένα heartbeat (γεγονός που συμβαίνει σε τακτά χρονικά διαστήματα) και η μία οντότητα δεν έχει λάβει ένα μήνυμα ok από την άλλη ώστε να διαπιστώσει ότι λειτουργεί σωστά τότε η οντότητα στέλνει ένα μήνυμα στον supervisor ειδοποιώντας τον ότι δεν έχει λάβει κάποιο μήνυμα από την άλλη οντότητα. Αυτός με τη σειρά του στέλνει ένα μήνυμα στο λειτουργικό συστατικό να αναλάβει το ρόλο του πρωτεύοντος και συγχρόνως το μη λειτουργικό γίνεται δευτερεύον. Επιπλέον, ο supervisor έχει το δικαίωμα να επιδιορθώσει την οντότητα που υπέστη βλάβη (το γεγονός αυτό δηλώνεται στο Ocas ως reset). Σε περίπτωση που για κάποιο λόγο καταρρεύσει ο supervisor (failure) οι λειτουργίες του αυτές παύουν να υφίστανται και στο δίκτυο μπορεί να εμφανιστούν προβλήματα όπως συγκρούσεις τρένων ή εκτροχιασμοί.

3.3 Ανάλυση του κώδικα

Για την ανάπτυξη του μοντέλου στην AltaRica υλοποιήσαμε τους σταθμούς ως components ο καθένας από τους οποίους περικλείει δύο sub-components. Στο Cecilia Ocas ο σταθμός υλοποιήθηκε ως οντότητα τύπου Equipment (stathmos) και οι sub-components ως οντότητες τύπου Components (stationn). Ο λόγος ύπαρξης των δύο sub-components για το σταθμό είναι ότι σε περίπτωση που συμβεί κάποια βλάβη στο σύστημα και καταρρεύσει ο ένας component να υπάρχει ένας άλλος “αντίγραφο” αυτού με τον οποίο να επιτελεί τις ίδιες λειτουργίες και να αναλάβει τη συνέχιση της ορθής λειτουργίας του δικτύου εμποδίζοντας τυχούσες συγκρούσεις ή εκτροχιασμούς τρένων από το να συμβούν. Έτσι, αρχικά και όσο δεν υπάρχει βλάβη στο σταθμό το ένα συστατικό (stationn1) είναι σε κατάσταση active και ok δηλαδή παίζει το ρόλο του πρωτεύοντος και λειτουργεί σωστά, ενώ το άλλο (stationn2) είναι σε κατάσταση shadow και ok δηλαδή δευτερεύον και λειτουργεί επίσης ορθά. Με την πάροδο του χρόνου μπορεί να συμβεί κάποια βλάβη στο πρωτεύον συστατικό. Το γεγονός αυτό θα γίνει αντιληπτό

από τον δευτερεύον μόλις έρθει το προκαθορισμένο χρονικό διάστημα (heartbeat) στο οποίο οι δύο οντότητες ανταλλάσουν μηνύματα εφόσον λειτουργούν σωστά και το δευτερεύον δε λάβει κάποιο μήνυμα από τον πρωτεύων (αφού αυτό έχει υποστεί βλάβη). Τη στιγμή εκείνη το δευτερεύον στέλνει ένα μήνυμα στον supervisor δηλώνοντάς του ότι το πρωτεύον έχει υποστεί βλάβη. Τότε, ο supervisor ειδοποιεί το δευτερεύον να πάρει τη θέση του πρωτεύοντος ενώ ταυτόχρονα το πρωτεύον γίνεται δευτερεύον.

Ανάλογη υλοποίηση εφαρμόστηκε και για τα DSBs. Υπάρχει δηλαδή το Equipment DSB και τα Components του dsb2. Αρχικά, το dsb25 είναι active και ok, ενώ το dsb24 είναι shadow και ok.

Πρέπει να αναφέρουμε ότι η συγκεκριμένη υλοποίηση αφορά στη διαχείριση και ενός δεύτερου τρένου σε περίπτωση που αυτό θέλει να εισέλθει σε ένα κανάλι στο οποίο βρίσκεται ήδη μία οντότητα. Για τη διαχείριση περισσότερων τρένων από ένα σταθμό θα πρέπει να γίνει μία σχετική επέκταση του μοντέλου.

Αναλυτική περιγραφή της λειτουργίας του μοντέλου

Αρχικά, όσο δεν υπάρχουν τρένα τα οποία επιθυμούν πρόσβαση σε κάποιο κανάλι και όσο δεν υπάρχει βλάβη σε κάποια οντότητα, οι καταστάσεις τόσο των σταθμών όσο και DSBs και του supervisor παραμένουν αμετάβλητες. Αναλυτικότερα, η κατάσταση situation των stationn1 και stationn2 κάθε σταθμού έχει την τιμή ok δηλώνοντας την ορθή λειτουργία των οντοτήτων, η κατάσταση status του stationn1 κάθε σταθμού έχει την τιμή active ενώ η αντίστοιχη κατάσταση του stationn2 την τιμή shadow δηλώνοντας το πρωτεύον και το δευτερεύον συστατικό. Η κατάσταση statt κάθε stationn1 έχει την τιμή first δηλώνοντας ότι είναι active και ok ενώ κάθε stationn2 έχει την τιμή third δηλώνοντας ότι είναι shadow και ok. Η κατάσταση etat τόσο του stationn1 όσο και του stationn2 έχουν την τιμή no δηλώνοντας ότι δεν υπάρχει τρένο στον αντίστοιχο σταθμό και η κατάσταση var των stationn1 και stationn2 ,η οποία χρησιμοποιείται από το stathmos για την αποστολή μηνύματος στον αμέσως επόμενο σταθμό όταν φύγει ένα τρένο από αυτόν και ζητήσει πρόσβαση για το κανάλι που ελέγχει ο επόμενος, έχει την τιμή zero δηλώνοντας ότι δεν υπάρχει τρένο που αναχώρησε από το σταθμό και πηγαίνει προς τον επόμενο. Επίσης, οι stationn αναλαμβάνουν να ανταλλάσσουν, όπως και προαναφέρθηκε, ένα μήνυμα σε προκαθορισμένα χρονικά διαστήματα για να διαπιστωθεί η ορθή ή μη λειτουργία τους. Έτσι, αρχικά το μήνυμα που στέλνει ο stationn1 έχει

την τιμή ok ενώ ο stationn2 την τιμή okk υποδηλώνοντας την ορθή λειτουργία τους. Οι stationn δεν αποστέλλουν κάποιο μήνυμα σε άλλη οντότητα. Το Equipment του σταθμού (stathmos) αρχικοποιεί τις καταστάσεις των Components του και είναι αυτό που αναλαμβάνει την αποστολή μηνυμάτων στο DSB του σταθμού, στον supervisor και στον επόμενο σε σειρά σταθμό, όπως επίσης και δέχεται τα μηνύματα από τις προαναφερθείσες οντότητες. Αρχικά, η έξοδός του προς τον supervisor (Osuper) έχει την τιμή no όπως επίσης και η είσοδός του από αυτόν (Isuper) δηλώνοντας ότι τα δύο stationn λειτουργούν σωστά. Η έξοδος του προς το DSB (Odsb) έχει επίσης την τιμή no όπως και η είσοδός του από αυτό (Idsb). Επίσης, ο κάθε stathmos έχει δύο λογικές εξόδους προς τον επόμενό του (Onext, Onext2) και δύο λογικές εισόδους από τον προηγούμενό του (Ibefore, Ibefore2). Οι δύο είσοδοι και έξοδοι χρησιμοποιούνται για καθένα από τα δύο τρένα. Δηλαδή, αν από ένα σε ένα σταθμό υπάρχουν δύο τρένα μόλις φύγει το ένα ο σταθμός θα αλλάξει την τιμή της μίας εξόδου του και κατά συνέπεια θα αλλάξει και η τιμή της εισόδου από τον επόμενο σταθμό ενώ η άλλη έξοδος θα μείνει ανεπηρέαστη. Όταν φύγει και το δεύτερο τρένο που υπάρχει στο σταθμό θα αλλάξει η τιμή της δεύτερης εξόδου προς τον επόμενο. Αρχικά λοιπόν, οι δύο έξοδοι και είσοδοι έχουν την τιμή false. Παρόμοια το component του dsb2 εμπεριέχει την κατάσταση statuss η οποία αρχικά είναι για το dsb25 active και για το dsb24 shadow, την κατάσταση sit η οποία έχει την τιμή ok και για τις δύο οντότητες υποδηλώνοντας την ορθή λειτουργία τους, την κατάσταση stat η οποία είναι για το dsb25 first (active, ok) και για το dsb24 third (shadow, ok) και την κατάσταση etat η οποία είναι free δηλώνοντας ότι δεν υπάρχει τρένο στο κανάλι που ελέγχεται από το αντίστοιχο DSB και σταθμό και συνεπώς είναι ελεύθερο. Σημειώνουμε ότι την αρχικοποίηση των καταστάσεων statuss, sit, stat με τις τιμές που προαναφέραμε την αναλαμβάνει η οντότητα DSB. Τα Components dsb25 και dsb24 επειδή είναι αντίγραφα του Component dsb2 ορίζονται να έχουν τις ίδιες ακριβώς τιμές στις καταστάσεις τους και για το λόγω αυτό που διαμορφώνει τη διαφοροποίηση στις τιμές των μεταβλητών τους είναι το DSB. Το ανάλογο συμβαίνει και με τα stationn και το stathmos. Όσον αφορά στις μεταβλητές εισόδου – εξόδου η μόνη είσοδος και έξοδος από και προς τα Components είναι αυτές που ανταλλάσουν μεταξύ τους τα δύο συστατικά σε τακτά χρονικά διαστήματα. Για την υλοποίηση αυτού ορίσαμε το event heartbeat το οποίο ενεργοποιείται σε τακτά χρονικά διαστήματα και εξυπηρετεί την ανταλλαγή των μηνυμάτων. Η οντότητα του DSB όπως είπαμε είναι υπεύθυνη για την αρχικοποίηση των καταστάσεων των δύο συστατικών της όπως επίσης ευθύνεται για την ανταλλαγή μηνυμάτων με τον supervisor, το σταθμό και τα δύο γειτονικά DSB

(προηγούμενο, επόμενο). Αρχικά, η είσοδος και η έξοδος από και προς τον supervisor έχουν την τιμή no δηλώνοντας ότι δεν υπάρχει κάποια βλάβη σε κάποιο συστατικό. Επίσης, η είσοδος και η έξοδος από και προς το σταθμό έχουν επίσης την τιμή no δηλώνοντας ότι δεν υπάρχει κάποιο τρένο που επιθυμεί πρόσβαση στο κανάλι για το οποίο είναι υπεύθυνα το DSB και ο αντίστοιχος σταθμός του. Η έξοδος (Oafter) στο επόμενο DSB έχει την τιμή nothing δηλώνοντας ότι δεν υπάρχει τρένο στον επόμενο σταθμό ώστε να του ζητηθεί η άδεια για να εισέλθει στον επόμενο σταθμό, η είσοδός του από το επόμενο DSB (Iafter) είναι επίσης nothing δηλώνοντας ότι δεν υπάρχει κάποιο τρένο στον επόμενο σταθμό, η έξοδός του στο προηγούμενο DSB (Obefore) είναι πάλι nothing αφού δεν υπάρχει κάποιο τρένο στον αντίστοιχο σταθμό του και η είσοδός του από το προηγούμενο DSB (Ibefore) είναι κι αυτή nothing αφού ο προηγούμενος σταθμός δεν έχει κάποιο μήνυμα από αυτόν ότι έχει τρένο.

Όσον αφορά στην οντότητα του supervisor έχει δύο λογικές καταστάσεις. Η κατάσταση stat αρχικά έχει την τιμή true και η τιμή της αλλάζει όταν ο supervisor επιδιορθώσει (reset) κάποια οντότητα που έχει υποστεί βλάβη. Η κατάσταση etat είναι κι αυτή αρχικά true και αλλάζει τιμή μόλις συμβεί κάποια βλάβη στον ίδιο τον supervisor. Επίσης, οι είσοδοι και οι έξοδοι του από και προς τους σταθμούς και τα DSB έχουν την τιμή no αφού δεν υπάρχει κάποια βλάβη στο σύστημα.

Τώρα υποθέτουμε ότι φτάνει ένα τρένο σε ένα σταθμό για παράδειγμα στον σταθμό stathmos1. Το γεγονός της άφιξης ενός τρένου στο stathmos1 δηλώνεται στο Ocas ως SyncTrain1. Αντίστοιχα, τα γεγονότα SyncTrain2, SyncTrain3 και SyncTrain4 υποδηλώνουν την άφιξη ενός τρένου στους σταθμούς stathmos2, stathmos3, stathmos5 αντίστοιχα. Αμέσως τότε η κατάσταση των δύο συστατικών του stathmos1 (stationn1, stationn2) etat παίρνει την τιμή one δηλώνοντας ότι υπάρχει ένα τρένο στο σταθμό. Επίσης, μεταβάλλεται και η κατάσταση etat των dsb24, dsb25 του DSB2 σε block μπλοκάροντας έτσι το DSB από το να επιτρέψει την κίνηση κάποιου άλλου τρένου που επιθυμεί πρόσβαση στο συγκεκριμένο κανάλι που υπάρχει ήδη ένα άλλο. Η κατάσταση μεταβάλλεται λόγω του event train στο Component του dsb2. Το global event SyncTrain1 που προαναφέραμε υπάρχει λόγω του ότι θέλαμε να συγχρονίσουμε αρχικά τα event train στα stationn1 και stationn2 (πράγμα το οποίο γίνεται στο μενού Synchronizations του stathmos) όπως αντίστοιχα και τα event train στα dsb24 και dsb25 και έπειτα να συγχρονίσουμε μεταξύ τους τα δύο νέα event (trains, joinedtrain) που προέκυψαν από τον προηγούμενο συγχρονισμό. Όσον αφορά στις αλλαγές των με-

ταβλητών εισόδου – εξόδου παρατηρούμε ότι ο stathmos1 στέλνει ένα μήνυμα bit1 στο αντίστοιχο DSB2 το οποίο το δέχεται ως είσοδο, το DSB2 στέλνει στο προηγούμενο DSB1 μήνυμα bit1 δηλώνοντας ότι έφτασε τρένο, το DSB1 του απαντάει με ένα μήνυμα bit2 το οποίο λαμβάνει το DSB2 και το στέλνει στον σταθμό του επιτρέποντας με τη διαδικασία αυτή τη διέλευση του τρένου στο κανάλι. Αμέσως μετά την πυροδότηση του γεγονότος SyncTrain1 ενεργοποιείται το γεγονός SyncLeave1 μέσω του οποίου δηλώνουμε ότι το τρένο φεύγει από το σταθμό stathmos1 και κατευθύνεται στο stathmos5 (δεξιόστροφη κίνηση). Το SyncLeave1 είναι κι αυτό ένα global event. Για να το ορίσουμε αρχικά συγχρονίζουμε το event leave των stationn1 και stationn2 στο τμήμα Synchronizations του stathmos (με την ονομασία leaving), επίσης με τον ίδιο τρόπο συγχρονίζουμε το event leave στα dsb24 και dsb25 (με την ονομασία joinedleave). Έπειτα, στο global synchronizations (το οποίο είναι υπεύθυνο για τον ορισμό καθολικών συγχρονισμών στο μοντέλο), συγχρονίζουμε τα events joinedleave του DSB2 με το joinedtrain του DSB6, το leaving του stathmos1 και το trains του stathmos5. Ο συγχρονισμός όλων αυτών έγινε για να δηλώσουμε ότι μόλις φύγει το τρένο από το stathmos1 αυτό πηγαίνει στο stathmos5. Με τον τρόπο αυτό ενημερώνονται και οι σταθμοί και τα αντίστοιχα DSBs τους. Επίσης, το κάνουμε έτσι για να μη χρειάζεται να πυροδοτήσουμε ξεχωριστά events για την αναχώρηση ενός τρένου από ένα σταθμό και την άφιξή του στον επόμενο. Έτσι, η διαδικασία αυτή ενοποιείται με την πυροδότηση ενός μόνο γεγονότος. Για αντίστοιχους λόγους ορίζουμε και τα καθολικά γεγονότα SyncLeave2, SyncLeave3, SyncLeave4. Μόλις πυροδοτήσουμε το SyncLeave1 η κατάσταση etat των dsb2 του DSB2 γίνεται free αφού δεν υπάρχει πλέον τρένο στο stathmos1, ενώ μπλοκάρεται το DSB6. Η κατάσταση των δύο συστατικών του stathmos1 etat γίνεται no αφού δεν υπάρχει πλέον τρένο στο σταθμό, ενώ η κατάσταση var γίνεται one προκειμένου να εξυπηρετήσει την αποστολή μηνύματος από το σταθμό προς τον επόμενό του για να τον ειδοποιήσει ένα τρένο κατευθύνεται προς εκείνον. Επίσης, η κατάσταση etat του stathmos5 γίνεται one δηλώνοντας ότι υπάρχει ένα τρένο σε αυτόν. Η είσοδος του DSB1 (DSB προηγούμενου σταθμού από από τον οποίο αναχώρησε το τρένο) από το DSB2 γίνεται τώρα nothing όπως και η έξοδος του DSB1 προς το DSB2. Η έξοδος του stathmos1 (Onext) προς το stathmos5 γίνεται true. Ακόμη, η έξοδος του stathmos1 προς το DSB του επανέρχεται στην κατάσταση no ενώ ο επόμενος σταθμός στέλνει bit1 στο αντίστοιχο DSB του ενημερώνοντάς το για την άφιξη του τρένου. Το DSB6 του σταθμού που υποδέχεται το τρένο στέλνει με τη σειρά του bit1 στο DSB του προηγούμενου σταθμού και έπειτα αυτός του απαντάει με ένα bit2 το οποίο λαμβάνει και στέλνει στο

σταθμό προκειμένου να επιτρέψει την είσοδο του τρένου στο κανάλι. Η ίδια διαδικασία ακολουθείται, όπως καταλαβαίνουμε εάν βάλουμε ένα τρένο σε οποιονδήποτε από τους τέσσερεις σταθμούς και ενεργοποιήσουμε την κίνησή του στο δίκτυο.

Υπάρχει η περίπτωση να φτάσει σε ένα σταθμό κι ένα δεύτερο τρένο. Θεωρούμε ότι αυτό συμβαίνει στο σταθμό stathmos1. Σε αυτήν την περίπτωση η κατάσταση etat των δύο συστατικών του DSB2 παίρνει την τιμή block2 δηλώνοντας ότι δύο τρένα υπάρχουν στο σταθμό. Επίσης, η κατάσταση etat των stationn1, stationn2 του σταθμού παίρνει την τιμή two αφού έφτασε κι ένα δεύτερο τρένο. Οι υπόλοιπες καταστάσεις παραμένουν ως έχουν. Όσον αφορά στις μεταβλητές εισόδου – εξόδου όλες όσες είχαν τροποποιηθεί λόγω του πρώτου τρένου επανέρχονται στην αρχική τους κατάσταση ούτως ώστε να μη γίνει ανταλλαγή μηνυμάτων άφιξης νέου τρένου μεταξύ των σταθμών και των DSBs και μπορέσουν και τα δύο τρένα να φύγουν παράλληλα από το σταθμό και επέλθει σύγκρουση στο κανάλι. Τώρα λοιπόν αφού έχει ενεργοποιηθεί από πριν (την άφιξη του πρώτου τρένου) το γεγονός SyncLeave1 δίνεται η δυνατότητα στο ένα τρένο να φύγει από το σταθμό και να κατευθυνθεί προς τον επόμενο. Επιπλέον, οι καταστάσεις και οι είσοδοι και έξοδοι των συστατικών των σταθμών και των DSBs γίνονται ακριβώς όπως αυτές που περιγράψαμε στην περίπτωση άφιξης ενός μόνο τρένου στο σταθμό με τη διαφορά ότι η μεταβλητή var στο σταθμό τώρα έχει την τιμή two εξυπηρετώντας την αποστολή μηνύματος στον επόμενο σταθμό δηλώνοντας ότι το δεύτερο τρένο φτάνει σε αυτόν. Αμέσως μετά ενεργοποιείται το καθολικό γεγονός SyncLeave 4 για την αναχώρηση του τρένου που μόλις έφτασε στο το stathmos5. Παρατηρούμε ότι μέχρι να φύγει το τρένο από αυτόν το σταθμό το τρένο που βρίσκεται ακόμη στο σταθμό stathmos1 δε μπορεί να φύγει από αυτόν αφού όπως είπαμε κάθε φορά μόνο ένα τρένο πρέπει να υπάρχει στο κανάλι. Αφού πυροδοτήσουμε και το γεγονός SyncLeave4 ενεργοποιείται και πάλι το γεγονός SyncLeave1 ούτως ώστε να μπορεί να αναχωρήσει και το άλλο τρένο από το σταθμό 1. Επίσης, παρατηρούμε ότι πλέον έχουν ενεργοποιηθεί και τα γεγονότα SyncTrains1 και SyncTrains3 αφού οι δύο σταθμοί έχουν από ένα τρένο και υπάρχει η δυνατότητα άφιξης ενός νέου σε αυτά (όχι όμως και αναχώρησης).

Ακόμη, κάποια στιγμή μπορεί να δημιουργηθεί κάποια βλάβη σε μία οντότητα του συστήματος. Αν δημιουργηθεί κάποια βλάβη σε κάποιο συστατικό τύπου stationn ή τύπου dsb2 αυτόματα η κατάσταση του από ok μετατρέπεται σε notok και οι τιμές του μεταβλητών εξόδου του συστατικού επανέρχονται στην αρχική κατάσταση (όπου δεν

υπάρχει τρένο σταθμό) εάν βέβαια δεν είναι ήδη σε αυτήν, αυτό συμβαίνει διότι όπως ορίσαμε στον κώδικα η διακίνηση των σωστών μηνυμάτων γίνεται εφόσον δεν υπάρχει βλάβη κάπου στο σύστημα. Αμέσως μόλις συμβεί το καθολικό γεγονός Joinedheartbeats στο stathmos1 το ένα συστατικό που λειτουργεί σωστά αντιλαμβάνεται τη βλάβη του άλλου αφού δεν παίρνει μήνυμα ok από αυτό. Για τη δυσλειτουργία ενός συστατικού ορίζεται τυπικά η έξοδος του μη λειτουργικού συστατικού να έχει την τιμή notok. Τότε, το λειτουργικό συστατικό στέλνει στον supervisor ένα μήνυμα msgs ειδοποιώντας τον για την βλάβη. Ο supervisor με τη σειρά του στέλνει ένα μήνυμα msg τυπικά και στα δύο συστατικά και έτσι το λειτουργικό συστατικό που έπαιζε το ρόλο του shadow component γίνεται τώρα active, ενώ το active που υπέστη βλάβη γίνεται shadow. Μέσω δηλαδή του supervisor η ορθή λειτουργία του δικτύου συνεχίζεται και επανέρχονται όλες οι καταστάσεις και οι μεταβλητές εισόδου – εξόδου στις τιμές που θα πρέπει να είναι και θα ήταν εάν δεν υπήρχε βλάβη. Δηλαδή, εάν υπήρχε κάποιο τρένο στο σταθμό που υπέστη βλάβη οι είσοδοι και έξοδοι από και προς τις οντότητες που σχετίζονται αποκτούν τις τιμές που πρέπει. Η μόνη αλλαγή είπαμε ότι συμβαίνει με τους ρόλους των δύο συστατικών. Η συνέχιση της ορθής λειτουργίας του δικτύου που περιγράψαμε γίνεται αμέσως μόλις συμβεί το επόμενο Joinedheartbeats στο σταθμό 1 που υποθέσαμε ότι δημιουργήθηκε κάποια βλάβη. Ο supervisor, όπως προαναφέρθηκε, έχει τη δυνατότητα να επιδιορθώνει βλάβη που τυχόν προκύπτει σε κάποιο συστατικό. Με λίγα λόγια, όσον αφόρα στη βλάβη που υποθέσαμε προηγουμένως, μπορούμε να τη διορθώσουμε με την πυροδότηση του γεγονότος reset που συμβαίνει μόνο στον supervisor. Έτσι, αυτός στέλνει ένα μήνυμα change στο σταθμό και στο επόμενο Joinedheartbeats του σταθμού παρατηρούμε ότι η κατάσταση του δυσλειτουργικού συστατικού είναι τώρα ok και οι είσοδοι και έξοδοι του συστατικού με τον supervisor επανέρχονται στην αρχική φυσιολογική κατάσταση.

Στο σύστημα έχουμε ορίσει ακόμη ένα Component με την ονομασία unexpected. Αυτό έχει ως εισόδους τις τιμές των εξόδων Onext, Onext2 που έχει κάθε σταθμός για να στέλνει στον επόμενο μήνυμα άφιξης ενός τρένου, και μία μόνο έξοδο (alarm) η οποία έχει την τιμή true εάν σε κάθε κανάλι του δικτύου υπάρχει το πολύ ένα τρένο, σε διαφορετική περίπτωση η τιμή του γίνεται false και τότε υπάρχει πρόβλημα στο δίκτυο. Ο τρόπος που υλοποιήσαμε το μοντέλο διατηρεί την τιμή της εξόδου αυτής πάντοτε ίση με true διατηρώντας το μοντέλο μας ασφαλές σε βλάβες. Στο Ocas υλοποιήσαμε τη σύνδεση του unexpected με τους σταθμούς στο μενού των System links όπου μας δίνεται η δυνατότητα να ορίσουμε συνδέσεις μεταξύ συστατικών του συστήματος.

Όσον αφορά στον τρόπο διαμόρφωσης των σταθμών και των DSBs θα μπορούσε κάποιος ίσως να ισχυριστεί γιατί δεν υλοποιήσαμε τα δύο ¨αντίγραφα¨ ως ξεχωριστές οντότητες και τα δημιουργήσαμε σαν sub-nodes ενός node. Θα περιγράψουμε τη συλλογική πορεία που ακολουθήσαμε για την υλοποίηση που προτείνουμε.

Ας υποθέσουμε ότι υλοποιούμε δύο ξεχωριστά components για το σταθμό. Τότε, δε χρειάζονται να υπάρχουν οι ρόλοι του πρωτεύοντος και του δευτερεύοντος συστατικού αφού και οι δύο οντότητες θα δρουν παράλληλα. Δηλαδή, θα έχουν ακριβώς τον ίδιο τρόπο εκτέλεσης λειτουργίας – όλες οι διαδικασίες θα γίνονται δύο φορές και όσον αφορά στις εισόδους και εξόδους και τις ανταλλαγές μηνυμάτων μεταξύ σταθμών, σταθμών – DSBs και αυτών των δύο με τον supervisor θα πρέπει για παράδειγμα, και τα δύο συστατικά που υλοποιούν τη λειτουργία του σταθμού να στέλνουν ταυτόχρονα μηνύματα στο DSB και αυτό θα λαμβάνει με τη σειρά του δύο εισόδους από το σταθμό. Σε περίπτωση που όλα πάνε καλά στο δίκτυο οι τιμές των μεταβλητών των δύο αυτών συστατικών θα είναι ακριβώς ίδιες, σε περίπτωση όμως που συμβεί κάποια βλάβη, η μεταβλητές δε θα είναι ίδιες και τα υπόλοιπα συστατικά που συνδέονται θα έχουν διαφορετικές εισόδους. Τί θα συμβεί σε αυτήν την περίπτωση στο σύστημα; Πώς θα συνεχιστεί η ορθή λειτουργία του δικτύου; Επίσης, ο supervisor δε θα χρειαζόταν να ειδοποιεί τη λειτουργική οντότητα να αλλάξει ρόλο με τη μη λειτουργική (αφού όπως είπαμε δε θα υπήρχε διάκριση ρόλων), το μόνο που θα μπορούσε να κάνει θα ήταν επιδιόρθωση ενός συστατικού που υπέστη βλάβη.

Η παραπάνω “εναλλακτική” διαδικασία που περιγράφει μία άλλη προσέγγιση ανάπτυξης του μοντέλου (όχι αυτήν που εφαρμόσαμε) δεν είναι η λογική του fail-safe modeling που επιδιώκουμε και επιπλέον δεν είναι πρακτική σε πραγματικές εφαρμογές, σε αντίθεση με αυτήν που υλοποιήσαμε. Σκοπός είναι να δημιουργήσουμε μία οντότητα η οποία θα λειτουργεί ¨σιωπηλά ¨ παράλληλα με την ενεργή. Θα εκτελεί ακριβώς τις ίδιες λειτουργίες με την ενεργή αλλά η ενεργή θα είναι αυτή που θα εκτελεί όλες τις ανταλλαγές μηνυμάτων με τις υπόλοιπες οντότητες. Το ¨αντίγραφό¨ της θα γίνεται ενεργό όταν αυτή υποστεί βλάβη. Αυτό το επιτυγχάνουμε μόνον με τη μοντελοποίηση που εφαρμόσαμε για το δίκτυό μας (δηλαδή ένα component συντίθεται από δύο subcomponents).

Οι συνδέσεις μεταξύ των σταθμών που ορίσαμε στο μοντέλο μας, εξυπηρετούν στην υλοποίηση του συστατικού unexpected μέσω του οποίου διαπιστώνουμε εάν το

μοντέλο μας παρουσιάζει λάθη. Επίσης, αυτό χρησιμεύει για τη δημιουργία fault tree για το μοντέλο μας.

4 Αποτελέσματα του μοντέλου και επεκτάσεις

4.1 Αποτελέσματα ανάλυσης

Δοκιμάσαμε αρκετά σενάρια για να διαπιστώσουμε εάν το μοντέλο μας λειτουργεί σωστά και αν παρεμποδίζεται το ενδεχόμενο ύπαρξης δύο τρένων στο ίδιο κανάλι. Όλα τα πειράματα έδειξαν ότι το δίκτυό μας εμφανίζει την αρμόζουσα συμπεριφορά σε κάθε σενάριο. Μπορεί κάποιος να παρακολουθήσει τη πορεία που εξελίσσεται η λειτουργία του δικτύου πυροδοτώντας διάφορα γεγονότα και παρακολουθώντας τον τρόπο με τον οποίο διαμορφώνονται οι μεταβλητές στο Debug View του εργαλείου Ocas. Τα αποτελέσματα της ανάλυσης τα πήραμε από τον step-by-step simulator του Cecilia Ocas. Επίσης, αξίζει να τονίσουμε ότι η υλοποίηση πραγματοποιήθηκε με όσο το δυνατόν λιγότερο πολύπλοκο και συγχρόνως λειτουργικό τρόπο. Λειτουργεί και σε συστήματα με μικρές απαιτήσεις μνήμης εξασφαλίζοντας συγχρόνως καλό χρόνο απόκρισης. Ένα μειονέκτημα του μοντέλου είναι ότι παρουσιάζει αρκετά μεγάλο χώρο καταστάσεων.

4.2 Επεκτάσεις

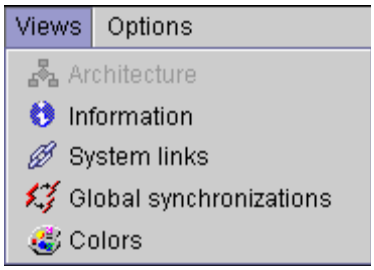
Όπως και αναφέραμε στην αρχή της εργασίας, το μοντέλο μας λειτουργεί για τον έλεγχο δρομολόγησης το πολύ δύο τρένων στο σταθμό. Θα μπορούσαμε να επεκτείνουμε το μοντέλο μας ώστε να ρυθμίζει τη δρομολόγηση περισσοτέρων τρένων που φτάνουν σε ένα σταθμό. Αυτό θα μπορούσε μάλλον να επιτευχθεί με την προσθήκη περισσοτέρων γεγονότων και ροών στο δίκτυο. Επίσης, θα μπορούσαμε να ελαχιστοποιήσουμε το χώρο καταστάσεων. Ακόμη, όπως προαναφέρθηκε τα αποτελέσματα τα πήραμε μόνον από τον step=by-step simulator και δεν ΄έχει γίνει model checking και πλήρης έλεγχος με FTA και δεν έχουν υπολογιστεί πλήρως τα min cut sets της αρχιτεκτονι-

κής. Για να γίνει έλεγχος FTA θα πρέπει το μοντέλο να τροποποιηθεί διότι η μορφή αυτή δεν καθιστά δυνατή τη fault tree analysis. Οπότε θα πρέπει να εφαρμοστούν επιπλέον έλεγχοι στο μοντέλο με τα εργαλεία Mec 5 και Arbor. Τέλος, θα μπορούσαμε να μοντελοποιήσουμε μία περισσότερο σύνθετη τοπολογία δικτύου, όπως για παράδειγμα αυτήν του μετρό της Αθήνας.

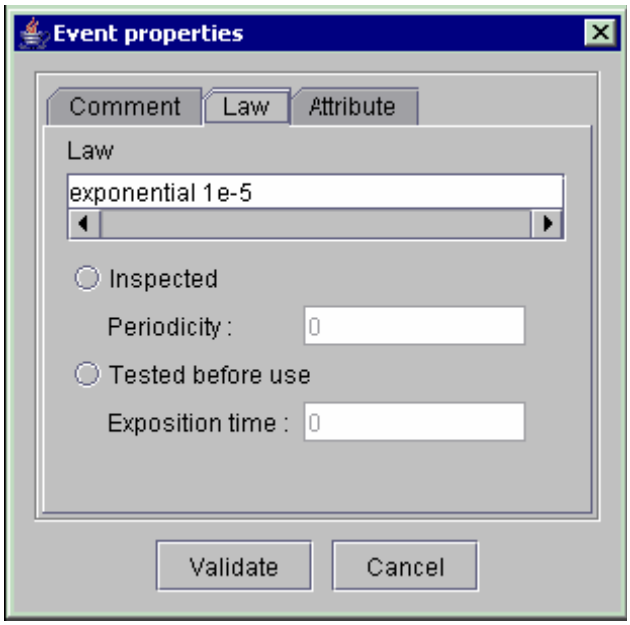
Βιβλιογραφία

[1] DASSAULT AVIATION. CECILIAWORKSHOP. Ocas Module. System Design And Analysis. User's Manual Ocas V4.3 (September 2007).
[2] Project IST – 004033.ASSERT. Automated proof based System and Software Engineering for Real-Time Applications. Author: Onera.
[3] A Timed Extension for AltaRica. Franc Cassez, Claire Pagetti and Olivier Roux.
[4] Interlocking control by Distributed Signal Boxes : design and verification with the SPIN model checker. Stylianos Basagiannis Panagiotis Katsaros Andrew Pombortsis. Department of Informatics, Aristotle University of Thessaloniki.
[5] Fail-safe network interlocking control by Distributed Signal Boxes. Stylianos Basagiannis Panagiotis Katsaros Andrew Pombortsis. Department of Informatics, Aristotle University of Thessaloniki
[6] Mec 5 Model-Checker. Alain Griffault and Aymeric Vincent. LaBRI, Bordeaux University.
[7] The grammar of the AltaRica language and its syntactic tree. Version X.XX. A. Griffault, G.Point, A.Vincent. March 15, 2006.
[8] Model-Based Safety Analysis Final Report. Anjali Joshi, Mike Whalen. Mats P.E. Heimdahl. Department of Computer Science and Engineering. Univesity of Minnesota.
[9] Model Checking. A Tutorial Introduction. Seminar: Sicherheitskritische Systeme Model checking--Eine Einfóhrung. Bearbeiter: Yuguo Sun.
[10] Vers la génération de modèles de sûreté de fonctionnement. Xavier Dumas, Claire Pagetti, Laurent Sagaspe, Pierre Bieber, and Philippe Dhaussy.
[11] SPIN Beginners' Tutorial. Theo C. Ruys. SPIN 2002 Workshop.
[12]Model Checking: A Brief. Introduction. Niklas Elmqvist. 8th February 2002.
[13] SAFETY ASSESSMENT WITH ALTARICA. Lessons learnt based on two aircraft system studies . Pierre Bieber, Christian Bougnol, Charles Castel, Jean-Pierre Heckmann Christophe Kehren, Sylvain Metge and Christel Seguin.
[14] Conception et validation d' un protocole avec le modèle ALTARICA. Alain Griffault. LaBRI (UMR CNRS 5800), Univrsité Bordeaux I 351.
[15] The AltaRica Formalism for Describing Concurrent Systems. André Arnold, Gérald Point, Alain Griffault and Antoine Rauzy. LaBRI, Université Bordeaux I and CNRS (UMR 5800).
[16] Wikipedia

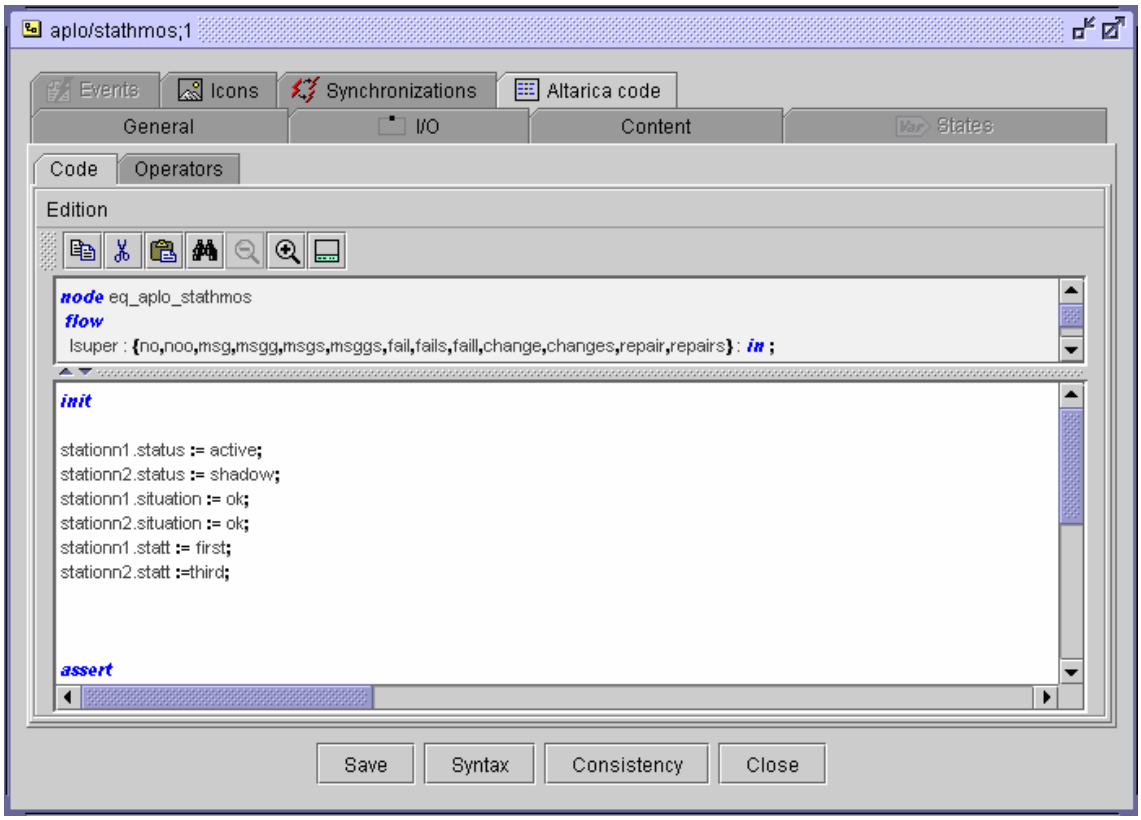
Παράρτημα



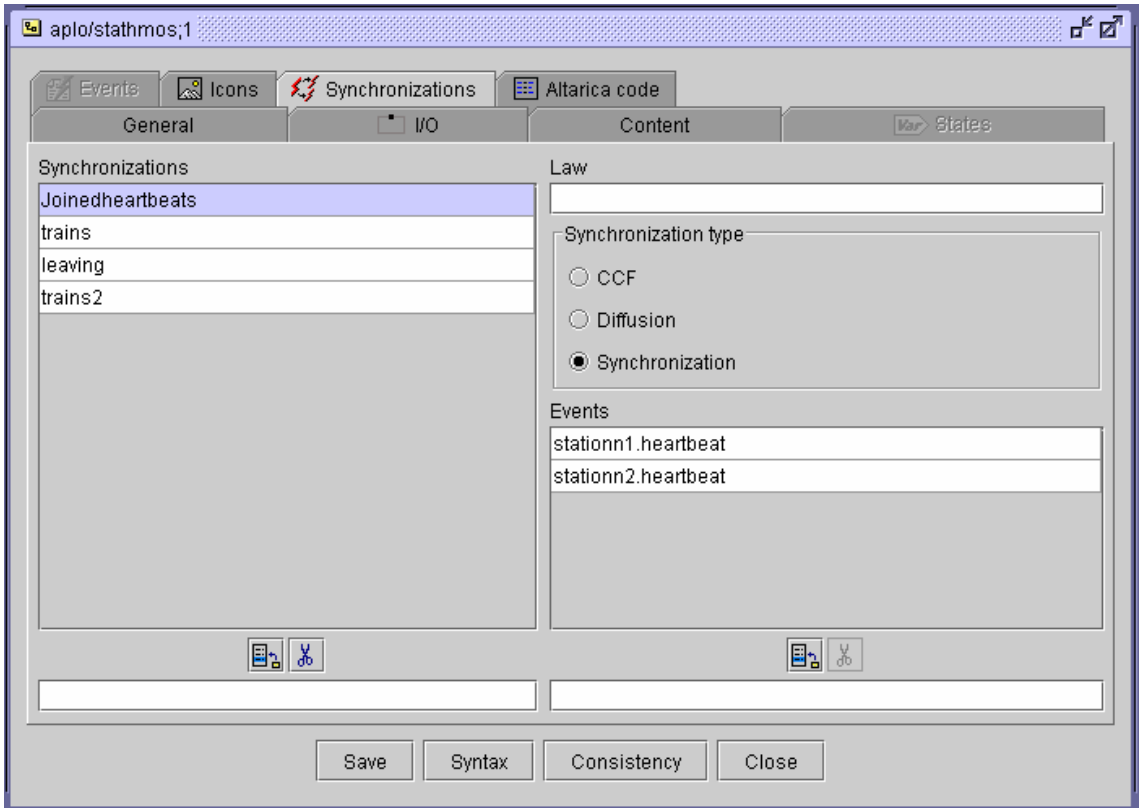
Εικόνα : στο μοντέλο μας χρησιμοποιήσαμε τα μενού system links για τις συνδέσεις με το συστατικό unexpected και το Global synchronizations για να ορίσουμε τους καθολικούς συγχρονισμούς του συστήματος.



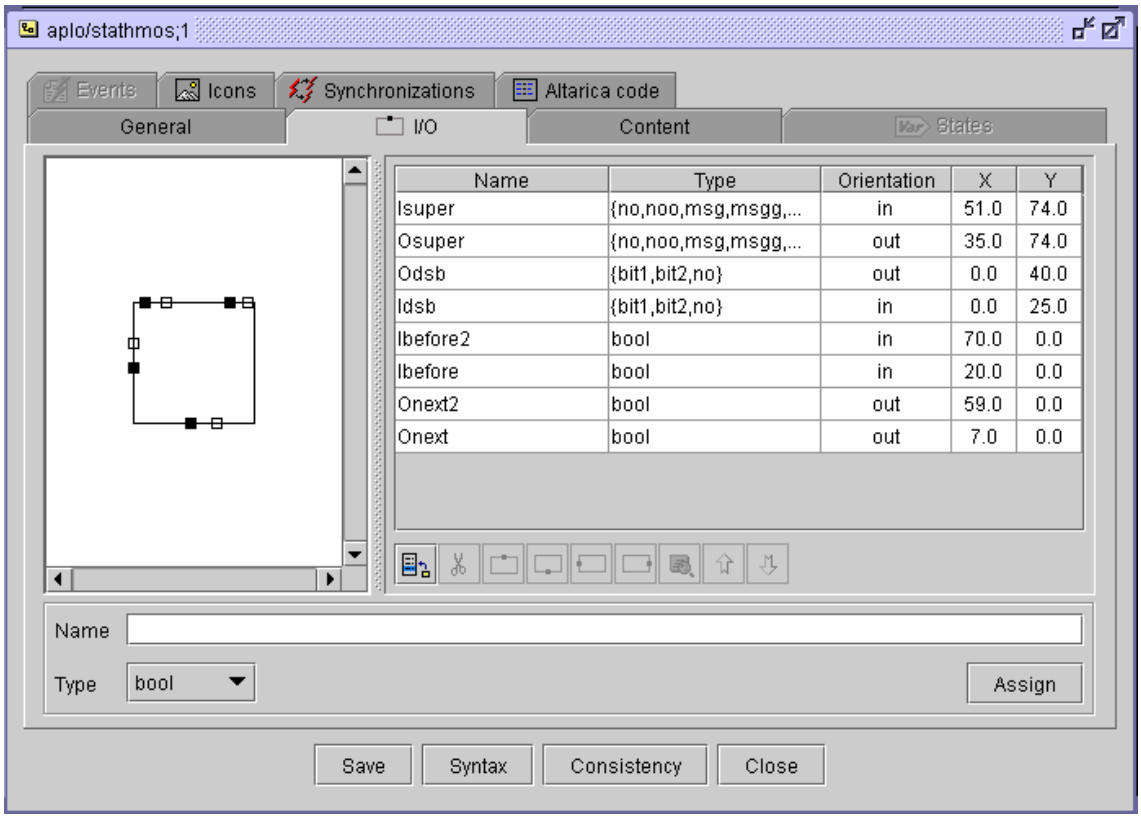
Εικόνα : με το μενού αυτό ρυθμίζουμε τη συχνότητα εμφάνισης ενός γεγονότος. Είναι προαιρετικό διότι αν θέλουμε μπορούμε να πυροδοτούμε τα γεγονότα χειροκίνητα όποτε το επιθυμούμε.



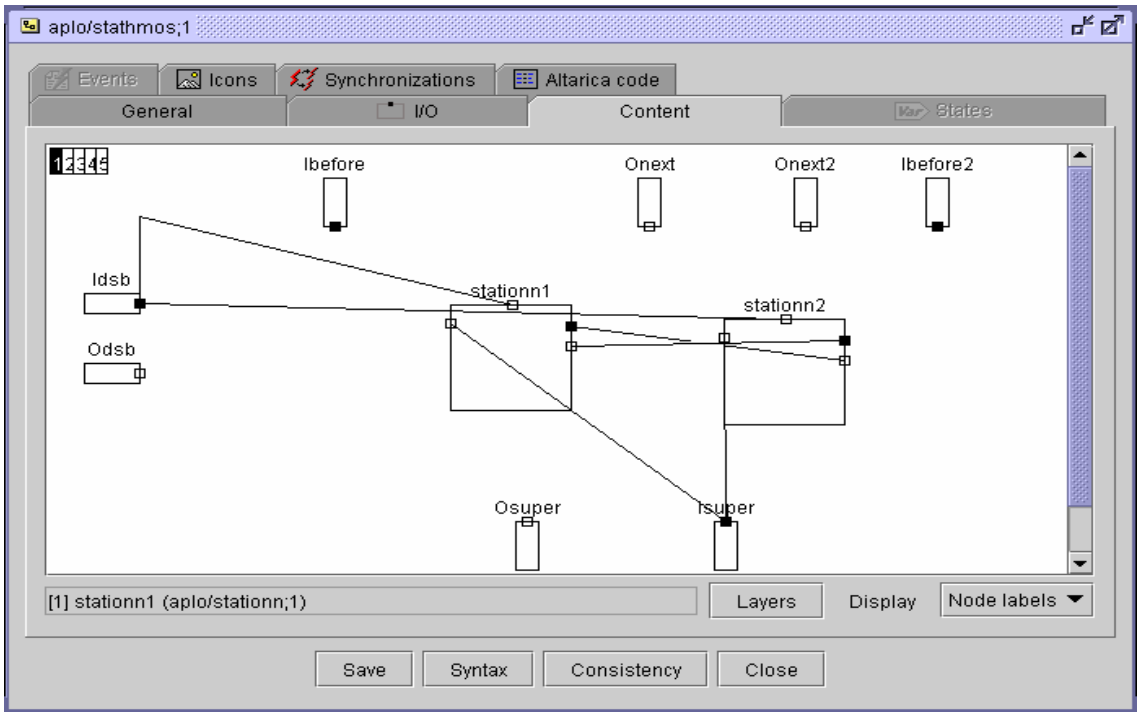
Εικόνα : κώδικας για την υλοποίηση του σταθμού



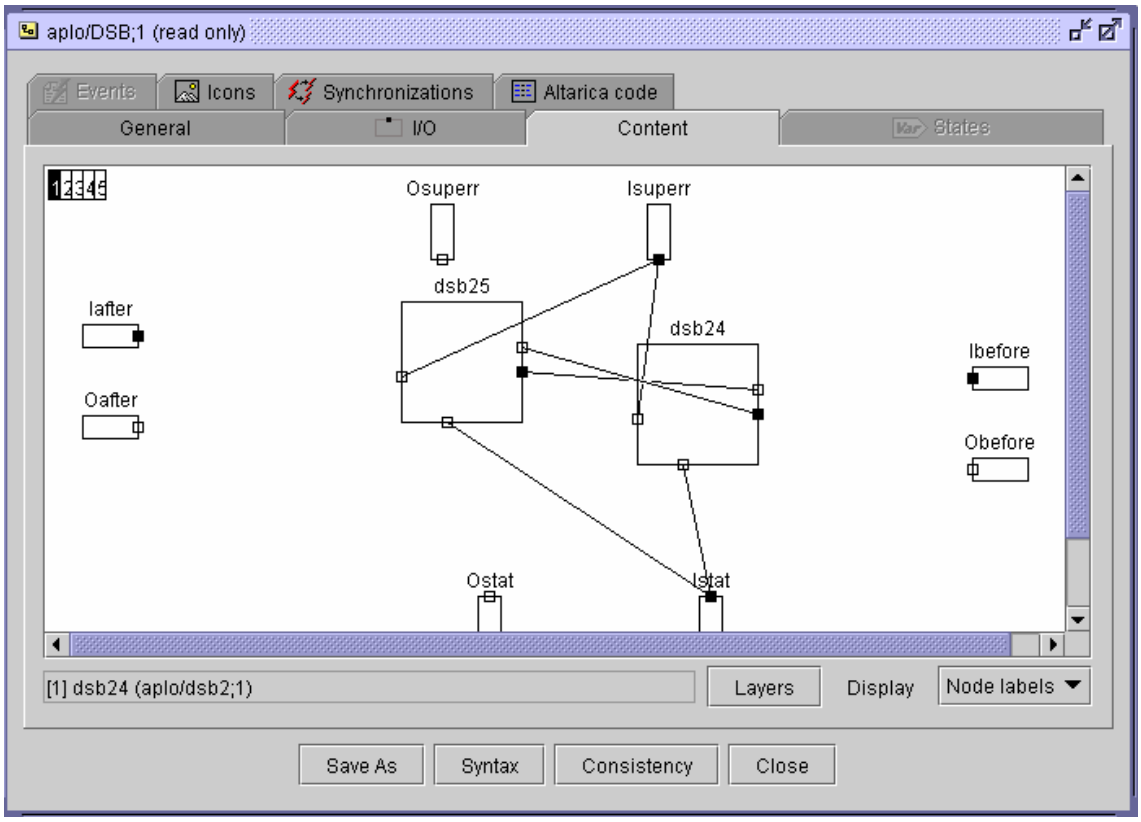
Εικόνα : το μενού Synchronizations



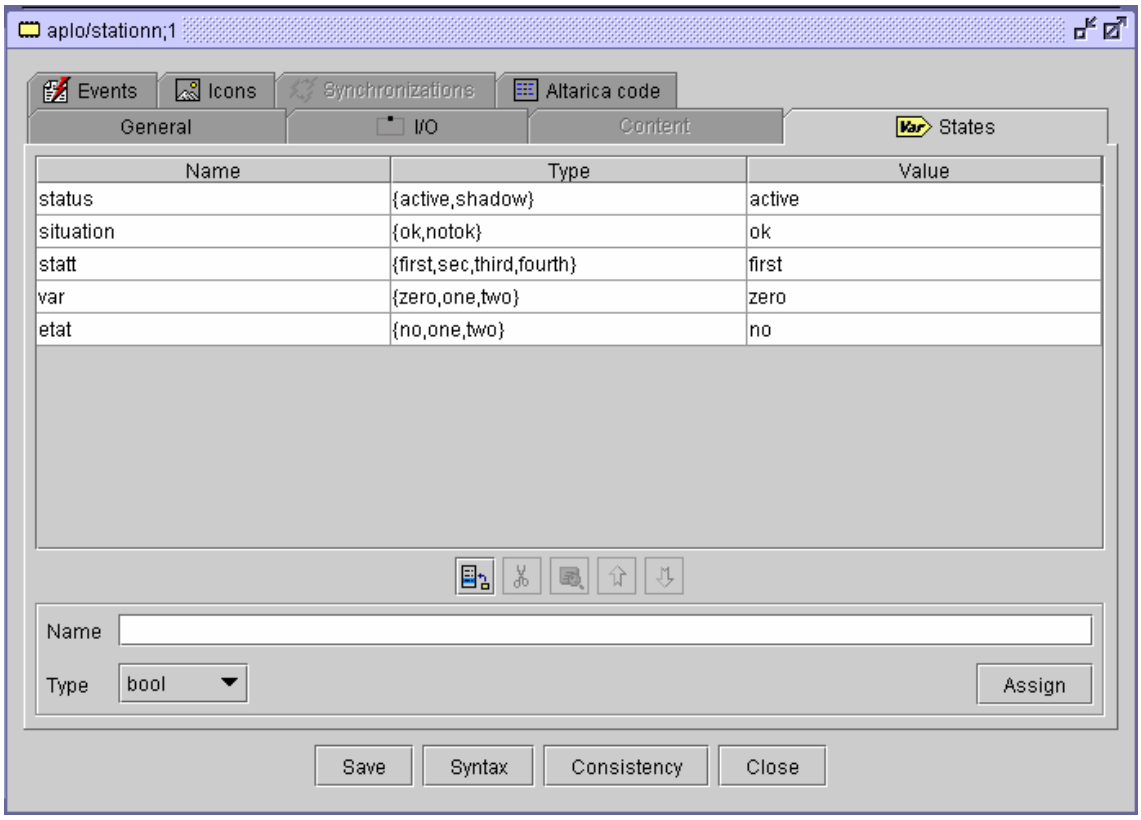
Εικόνα : μενού για δήλωση εισόδων – εξόδων



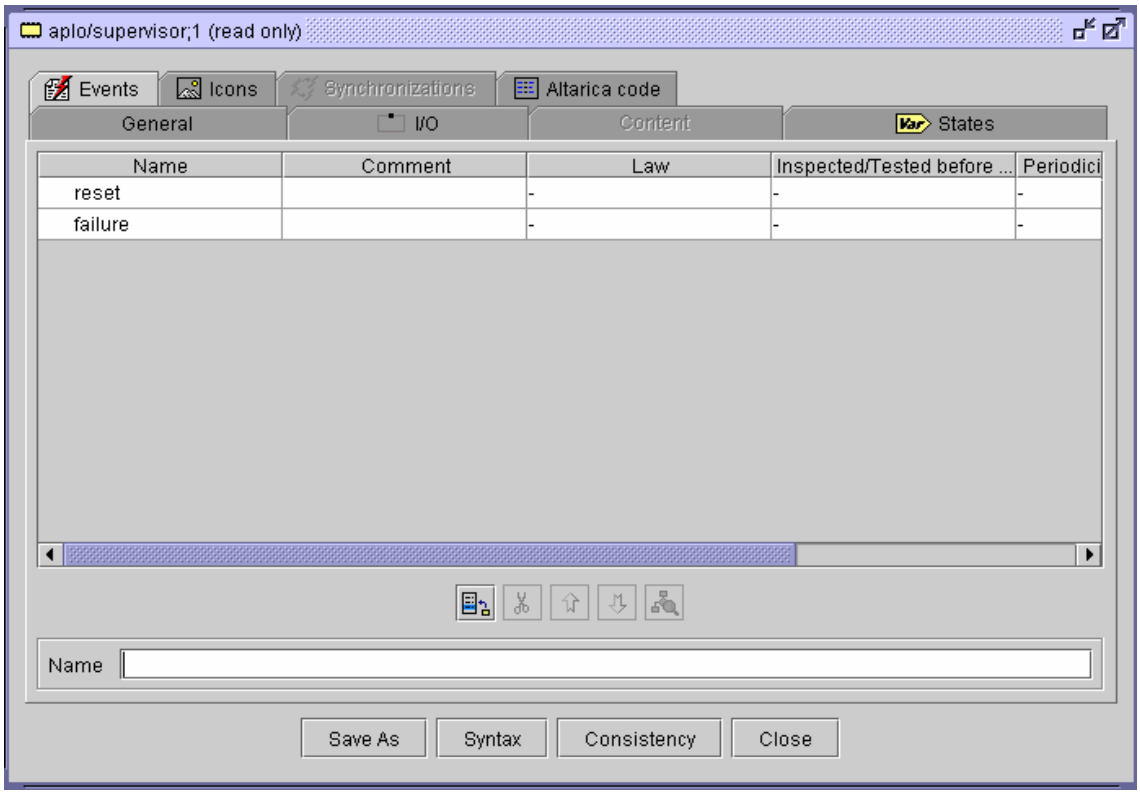
Εικόνα : Περιεχόμενο του σταθμού – τα δύο συστατικά που εμπεριέχει και η σύνδεση με αυτά



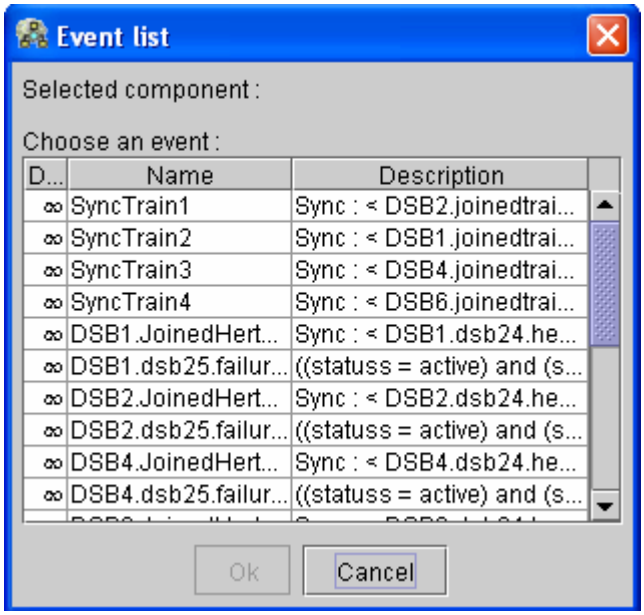
Εικόνα : Περιεχόμενο του DSB – τα δύο συστατικά που εμπεριέχει και η σύνδεσή του με αυτά



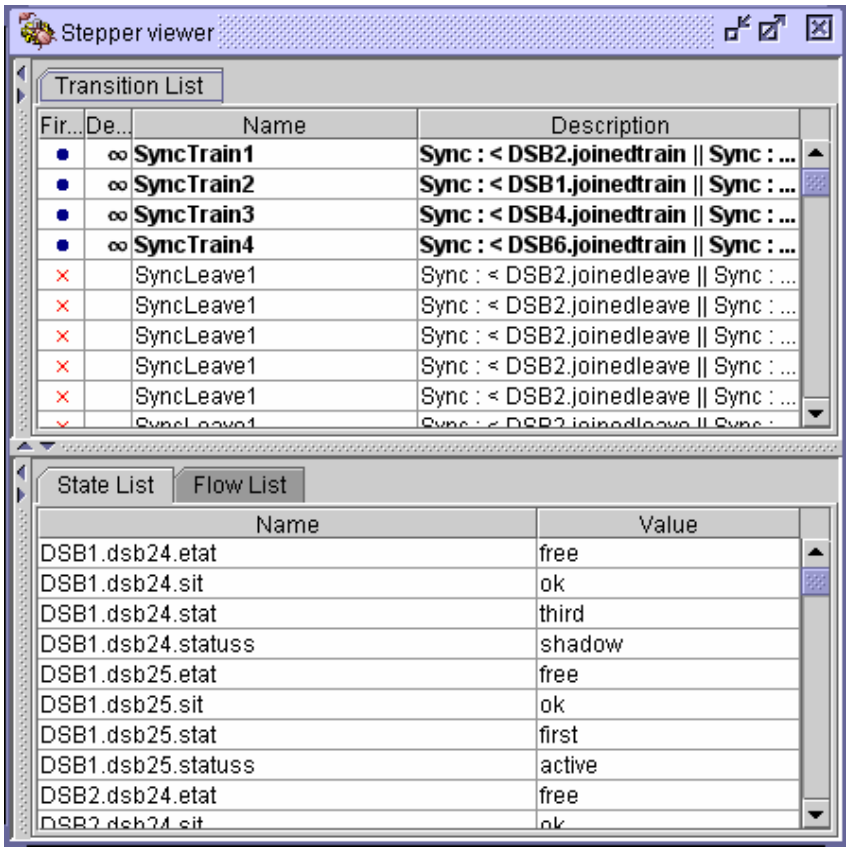
Εικόνα : δήλωση μεταβλητών κατάστασης του συστατικού stationn που περικλείεται στο σταθμό.



Εικόνα : Δήλωση γεγονότων στον supervisor – δεν έχουμε ορίσει συχνότητα εμφάνισης



Εικόνα : Λίστα γεγονότων που μπορούμε να πυροδοτήσουμε



Εικόνα : Παρουσιάζονται από το σύστημα οι τιμές των μεταβλητών κατάστασης και ροής για όλα τα συστατικά.

Κώδικας στην AltaRica

```
node aplo_unexpected
  flow
    I11:bool:in;
    I1:bool:in;
    I22:bool:in;
    I2:bool:in;
    I33:bool:in;
    I3:bool:in;
    I44:bool:in;
    I4:bool:in;
    alarm:bool:out;
  assert
    alarm = (if (((((I1 = true) and (I11 = true)) or ((I2 = true) and (I22 = true))) or ((I3 = true) and (I33 = true))) or ((I4 = true) and (I44 = true))) then true else false);
edon

node aplo_dsb2
  flow
    Istat:{bit1, bit2, no}:in;
    Iother:{notok, notokk, ok, okk}:in;
    Oother:{notok, notokk, ok, okk}:out;
    Isuperr:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;
  state
    sit:{notok, ok};
    statuss:{active, shadow};
    stat:{first, fourth, sec, third};
    etat:{block, block2, free};
```

```
event
  heartbeatt,
  failures,
  leave,
  train,
  train2;
trans
  (etat = free) |- train -> etat := block;
  (etat = block) |- leave -> etat := free;
  (etat = block) |- train2 -> etat := block2;
  (etat = block2) |- leave -> etat := block;
  ((statuss = active) and (sit = ok)) |- failures -> sit := notok;
  ((statuss = shadow) and (sit = notok)) |- failures -> sit := notok;
  ((statuss = active) and (sit = ok)) |- heartbeatt -> stat := first;
  ((statuss = active) and (sit = notok)) |- heartbeatt -> stat := sec;
  (stat = sec) |- heartbeatt -> statuss := shadow, stat := fourth;
  ((statuss = shadow) and (Isuperr = repair)) |- heartbeatt -> sit := ok, stat := third;
  ((statuss = active) and (Isuperr = repairs)) |- heartbeatt -> sit := ok, stat := first;
  (((statuss = shadow) and (sit = ok)) and (Isuperr = no)) |- heartbeatt -> stat := third;
  ((stat = third) and (Isuperr = msgs)) |- heartbeatt -> statuss := active, stat := first;
assert
  Oother = case {
    (stat = first) : ok,
    ((not (stat = first)) and (stat = sec)) : notok,
    ((not (stat = first)) and (not (stat = sec)) and (stat = third)) : okk,
    else notokk
  };
init
  sit := ok,
```

```
    statuss := active,
    stat := first,
    etat := free;
edon

node aplo_stationn
  flow
    Idsb:{bit1, bit2, no}:in;
    Isuper:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;
    Oallo:{notok, notokk, ok, okk}:out;
    Iallo:{notok, notokk, ok, okk}:in;
  state
    status:{active, shadow};
    situation:{notok, ok};
    statt:{first, fourth, sec, third};
    var:{one, two, zero};
    etat:{no, one, two};
  event
    heartbeat,
    failure,
    leave,
    train,
    train2;
  trans
    ((status = active) and (situation = ok)) |- failure -> situation := notok;
    ((status = shadow) and (situation = notok)) |- failure -> situation := notok;
    (etat = no) |- train -> etat := one;
    (etat = one) |- leave -> etat := no, var := one;
    (etat = one) |- train2 -> etat := two;
```

```
    (etat = two) |- leave -> etat := one, var := two;
    ((status = active) and (situation = ok)) |- heartbeat -> statt := first;
    ((status = active) and (situation = notok)) |- heartbeat -> statt := sec;
    (statt = sec) |- heartbeat -> status := shadow, statt := fourth;
    ((status = shadow) and (Isuper = change)) |- heartbeat -> situation := ok, statt := third;
    ((status = active) and (Isuper = changes)) |- heartbeat -> situation := ok, statt := first;
    (((status = shadow) and (situation = ok)) and (Isuper = no)) |- heartbeat -> statt := third;
    ((statt = third) and (Isuper = msgs)) |- heartbeat -> status := active, statt := first;
  assert
    Oallo = case {
      (statt = first) : ok,
      ((not (statt = first)) and (statt = sec)) : notok,
      ((not (statt = first)) and (not (statt = sec)) and (statt = third)) : okk,
      else notokk
    };
  init
    status := active,
    situation := ok,
    statt := first,
    var := zero,
    etat := no;
edon

node aplo_supervisor
  flow
    Odsb3:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;
    Odsb2:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;
```

Odsb1:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Odsb:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Idsb2:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Idsb1:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Idsb3:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Idsb:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Ostation1:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Ostation2:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Ostation3:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Ostation:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Istation1:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Istation2:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Istation3:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Istation:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

state

stat:bool;

etat:bool;

event

reset,

failure;

trans

(etat = true) |- reset -> stat := false;

(etat = true) |- failure -> etat := false;

assert

Odsb3 = (if (((Idsb3 = no) and (stat = true)) and (etat = true)) then no else (if (((Idsb3 = msg) and (stat = true)) and (etat = true)) then msg else (if (((Idsb3 = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Idsb3 = fail) and (stat = true)) and (etat = true)) then fail else (if (((Idsb3 = fails) and (stat = true)) and (etat = true)) then fails else (if (((Idsb3 = msgs) and (stat = false)) and (etat = true)) then repairs else (if (((Idsb3 = msg) and (stat = false)) and (etat = true)) then repair else no)))))),

Odsb2 = (if (((Idsb2 = no) and (stat = true)) and (etat = true)) then no else (if (((Idsb2 = msg) and (stat = true)) and (etat = true)) then msg else (if (((Idsb2 = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Idsb2 = fail) and (stat = true)) and (etat = true)) then fail else (if (((Idsb2 = fails) and (stat = true)) and (etat = true)) then fails else (if (((Idsb2 = msgs) and (stat = false)) and (etat = true)) then repairs else (if (((Idsb2 = msg) and (stat = false)) and (etat = true)) then repair else no)))))),

Odsb1 = (if (((Idsb1 = no) and (stat = true)) and (etat = true)) then no else (if (((Idsb1 = msg) and (stat = true)) and (etat = true)) then msg else (if (((Idsb1 = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Idsb1 = fail) and (stat = true)) and (etat = true)) then fail else (if (((Idsb1 = fails) and (stat = true)) and (etat = true)) then fails else (if (((Idsb1 = msgs) and (stat = false)) and (etat = true)) then repairs else (if (((Idsb1 = msg) and (stat = false)) and (etat = true)) then repair else no)))))),

Odsb = (if (((Idsb = no) and (stat = true)) and (etat = true)) then no else (if (((Idsb = msg) and (stat = true)) and (etat = true)) then msg else (if (((Idsb = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Idsb = fail) and (stat = true)) and (etat = true)) then fail else (if (((Idsb = fails) and (stat = true)) and (etat = true)) then fails else (if (((Idsb = msgs) and (stat = false)) and (etat = true)) then repairs else (if (((Idsb = msg) and (stat = false)) and (etat = true)) then repair else no)))))),

Ostation1 = (if (((Istation1 = no) and (stat = true)) and (etat = true)) then no else (if (((Istation1 = msg) and (stat = true)) and (etat = true)) then msg else (if (((Istation1 = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Istation1 = fail) and (stat = true)) and (etat = true)) then fail else (if (((Istation1 = fails) and (stat = true)) and (etat = true)) then fails else (if (((Istation1 = msg) and (stat = false)) and (etat = true)) then change else (if (((Istation1 = msgs) and (stat = false)) and (etat = true)) then changes else no)))))),

Ostation2 = (if (((Istation2 = no) and (stat = true)) and (etat = true)) then no else (if (((Istation2 = msg) and (stat = true)) and (etat = true)) then msg else (if (((Istation2 = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Istation2 = fail) and (stat = true)) and (etat = true)) then fail else (if (((Istation2 = fails) and (stat = true)) and (etat = true)) then fails else (if (((Istation2 = msg) and (stat = false)) and (etat = true)) then change else (if (((Istation2 = msgs) and (stat = false)) and (etat = true)) then changes else no)))))),

Ostation3 = (if (((Istation3 = no) and (stat = true)) and (etat = true)) then no else (if (((Istation3 = msg) and (stat = true)) and (etat = true)) then msg else (if (((Istation3 = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Istation3 = fail) and (stat = true)) and (etat = true)) then fail else (if (((Istation3 = fails) and (stat = true)) and (etat = true)) then fails else (if (((Istation3 = msg) and (stat = false)) and (etat = true)) then change else (if (((Istation3 = msgs) and (stat = false)) and (etat = true)) then changes else no)))))),

Ostation = (if (((Istation = no) and (stat = true)) and (etat = true)) then no else (if (((Istation = msg) and (stat = true)) and (etat = true)) then msg else (if (((Istation = msgs) and (stat = true)) and (etat = true)) then msgs else (if (((Istation = fail) and (stat = true)) and (etat = true)) then fail else (if (((Istation = fails) and (stat = true)) and (etat = true)) then fails else (if (((Istation = msg) and (stat = false)) and (etat = true)) then change else (if (((Istation = msgs) and (stat = false)) and (etat = true)) then changes else no)))))));

init

stat := true,

etat := true;

edon

node eq_aplo_DSB

flow

Istat:{bit1, bit2, no}:in;

Isuperr:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;

Osuperr:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;

Ostat:{bit1, bit2, no}:out;

Iafter:{bit1, nothing}:in;

Ibefore:{bit2, nothing}:in;

Obefore:{bit1, nothing}:out;

Oafter:{bit2, nothing}:out;

event

JoinedHertbeatt,

joinedtrain,

joinedleave,

joinedtrain2;

sub

dsb25:aplo_dsb2;

dsb24:aplo_dsb2;

assert

Osuperr = (if ((dsb25.stat = first) and (dsb24.stat = third)) then no else (if ((dsb25.stat = first) and (dsb24.stat = fourth)) then msg else (if ((dsb25.stat = sec) and (dsb24.stat = third)) then msgs else (if ((dsb25.stat = sec) and (dsb24.stat = fourth)) then fail else (if ((dsb25.stat = third) and (dsb24.stat = first)) then no else (if ((dsb25.stat = third) and (dsb24.stat = sec)) then msgs else (if ((dsb25.stat = fourth) and (dsb24.stat = first)) then msg else (if ((dsb25.stat = fourth) and (dsb24.stat = sec)) then fail else fails)))))))),

Ostat = (if ((((dsb25.etat = block) and (dsb25.statuss = active)) and (dsb25.sit = ok)) and (Ibefore = bit2)) then bit2 else (if ((((dsb24.etat = block) and (dsb24.statuss = active)) and (dsb24.sit = ok)) and (Ibefore = bit2)) then bit2 else no)),

Obefore = (if ((((dsb25.statuss = active) and (dsb25.sit = ok)) and (Istat = no)) and (dsb25.etat = free)) then nothing else (if ((((dsb25.statuss = active) and (dsb25.sit = ok)) and (Istat = bit1)) and (dsb25.etat = block)) then bit1 else (if ((((dsb24.statuss = active) and (dsb24.sit = ok)) and (Istat = no)) and (dsb24.etat = free)) then nothing else (if ((((dsb24.statuss = active) and (dsb24.sit = ok)) and (Istat = bit1)) and (dsb24.etat = block)) then bit1 else nothing)))),

Oafter = (if ((((dsb25.statuss = active) and (dsb25.sit = ok)) and (dsb25.etat = free)) and (Iafter = nothing)) then nothing else (if ((((dsb25.statuss = active) and (dsb25.sit = ok)) and (dsb25.etat = free)) and (Iafter = bit1)) then bit2 else (if ((((dsb24.statuss = active) and (dsb24.sit = ok)) and (dsb24.etat = free)) and (Iafter = nothing)) then nothing else (if ((((dsb24.statuss = active) and (dsb24.sit = ok)) and (dsb24.etat = free)) and (Iafter = bit1)) then bit2 else (if ((((dsb25.statuss = active) and (dsb25.sit = ok)) and (dsb25.etat = block)) and (Iafter = bit1)) then bit2 else (if ((((dsb24.statuss = active) and (dsb24.sit = ok)) and (dsb24.etat = block)) and (Iafter = bit1)) then bit2 else nothing))))))),

dsb25.Istat = Istat,

dsb25.Iother = dsb24.Oother,

dsb25.Isuperr = Isuperr,

dsb24.Istat = Istat,

dsb24.Iother = dsb25.Oother,

dsb24.Isuperr = Isuperr;

sync

<JoinedHertbeatt , dsb24.heartbeatt , dsb25.heartbeatt>,

<joinedtrain , dsb24.train , dsb25.train>,

<joinedleave , dsb24.leave , dsb25.leave>,

<joinedtrain2 , dsb24.train2 , dsb25.train2>;

init

dsb25.statuss := active,

dsb24.statuss := shadow,

dsb24.sit := ok,

dsb25.sit := ok,

```
    dsb25.stat := first,
    dsb24.stat := third;
  extern
    nodeproperty <global projectName> = "aplo/simple2";
    nodeproperty <global projectVersion> = "1";
    nodeproperty <global projectConfig> = "default";
    nodeproperty <global currentDate> = "2008-08-08 13:07:59";
edon

node eq_aplo_stathmos
  flow
    Isuper:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:in;
    Osuper:{change, changes, fail, faill, fails, msg, msgg, msggs, msgs, no, noo, repair, repairs}:out;
    Odsb:{bit1, bit2, no}:out;
    Idsb:{bit1, bit2, no}:in;
    Ibefore2:bool:in;
    Ibefore:bool:in;
    Onext2:bool:out;
    Onext:bool:out;
  event
    Joinedheartbeats,
    trains,
    leaving,
    trains2;
  sub
    stationn2:aplo_stationn;
    stationn1:aplo_stationn;
  assert
```

Osuper = (if ((stationn1.statt = first) and (stationn2.statt = third)) then no else (if ((stationn1.statt = first) and (stationn2.statt = fourth)) then msg else (if ((stationn1.statt = sec) and (stationn2.statt = third)) then msgs else (if ((stationn1.statt = sec) and (stationn2.statt = fourth)) then fail else (if ((stationn1.statt = third) and (stationn2.statt = first)) then no else (if ((stationn1.statt = third) and (stationn2.statt = sec)) then msgs else (if ((stationn1.statt = fourth) and (stationn2.statt = first)) then msg else (if ((stationn1.statt = fourth) and (stationn2.statt = sec)) then fail else fails)))))))),

Odsb = (if (((stationn1.status = active) and (stationn1.situation = ok)) and (stationn1.etat = no)) then no else (if (((stationn1.status = active) and (stationn1.situation = ok)) and (stationn1.etat = one)) then bit1 else (if (((stationn2.status = active) and (stationn2.situation = ok)) and (stationn2.etat = no)) then no else (if (((stationn2.status = active) and (stationn2.situation = ok)) and (stationn2.etat = one)) then bit1 else no)))),

Onext2 = (if (((stationn1.status = active) and (stationn1.situation = ok)) and (stationn1.var = two)) then true else (if (((stationn2.status = active) and (stationn2.situation = ok)) and (stationn2.var = two)) then true else false)),

Onext = (if (((stationn1.status = active) and (stationn1.situation = ok)) and (stationn1.var = one)) then true else (if (((stationn2.status = active) and (stationn2.situation = ok)) and (stationn2.var = one)) then true else false)),

stationn2.Idsb = Idsb,

stationn2.Isuper = Isuper,

stationn2.Iallo = stationn1.Oallo,

stationn1.Idsb = Idsb,

stationn1.Isuper = Isuper,

stationn1.Iallo = stationn2.Oallo;

sync

<Joinedheartbeats , stationn1.heartbeat , stationn2.heartbeat>,

<trains , stationn1.train , stationn2.train>,

<leaving , stationn1.leave , stationn2.leave>,

<trains2 , stationn1.train2 , stationn2.train2>;

init

stationn1.status := active,

```
    stationn2.status := shadow,
    stationn1.situation := ok,
    stationn2.situation := ok,
    stationn1.statt := first,
    stationn2.statt := third;
  extern
    nodeproperty <global projectName> = "aplo/simple2";
    nodeproperty <global projectVersion> = "1";
    nodeproperty <global projectConfig> = "default";
    nodeproperty <global currentDate> = "2008-08-08 13:07:59";
edon

node main
  event
    SyncTrain1,
    SyncTrain2,
    SyncTrain3,
    SyncTrain4,
    SyncLeave1,
    SyncLeave2,
    SyncLeave3,
    SyncLeave4,
    SyncTrains1,
    syncTrains2,
    SyncTrains3,
    SyncTrains4;
  sub
    unexpected1:aplo_unexpected;
    DSB6:eq_aplo_DSB;
```

```
  stathmos5:eq_aplo_stathmos;
  DSB4:eq_aplo_DSB;
  stathmos3:eq_aplo_stathmos;
  stathmos2:eq_aplo_stathmos;
  DSB1:eq_aplo_DSB;
  supervisor1:aplo_supervisor;
  DSB2:eq_aplo_DSB;
  stathmos1:eq_aplo_stathmos;
assert
  unexpected1.I11 = stathmos1.Onext2,
  unexpected1.I1 = stathmos1.Onext,
  unexpected1.I22 = stathmos5.Onext2,
  unexpected1.I2 = stathmos5.Onext,
  unexpected1.I33 = stathmos3.Onext2,
  unexpected1.I3 = stathmos3.Onext,
  unexpected1.I44 = stathmos2.Onext2,
  unexpected1.I4 = stathmos2.Onext,
  DSB6.Istat = stathmos5.Odsb,
  DSB6.Isuperr = supervisor1.Odsb3,
  DSB6.Iafter = DSB4.Obefore,
  DSB6.Ibefore = DSB2.Oafter,
  stathmos5.Isuper = supervisor1.Ostation3,
  stathmos5.Idsb = DSB6.Ostat,
  stathmos5.Ibefore2 = stathmos1.Onext2,
  stathmos5.Ibefore = stathmos1.Onext,
  DSB4.Istat = stathmos3.Odsb,
  DSB4.Isuperr = supervisor1.Odsb2,
  DSB4.Iafter = DSB1.Obefore,
  DSB4.Ibefore = DSB6.Oafter,
```

```
  stathmos3.Isuper = supervisor1.Ostation2,
  stathmos3.Idsb = DSB4.Ostat,
  stathmos3.Ibefore2 = stathmos5.Onext2,
  stathmos3.Ibefore = stathmos5.Onext,
  stathmos2.Isuper = supervisor1.Ostation1,
  stathmos2.Idsb = DSB1.Ostat,
  stathmos2.Ibefore2 = stathmos3.Onext2,
  stathmos2.Ibefore = stathmos3.Onext,
  DSB1.Istat = stathmos2.Odsb,
  DSB1.Isuperr = supervisor1.Odsb1,
  DSB1.Iafter = DSB2.Obefore,
  DSB1.Ibefore = DSB4.Oafter,
  supervisor1.Idsb2 = DSB4.Osuperr,
  supervisor1.Idsb1 = DSB1.Osuperr,
  supervisor1.Idsb3 = DSB6.Osuperr,
  supervisor1.Idsb = DSB2.Osuperr,
  supervisor1.Istation1 = stathmos2.Osuper,
  supervisor1.Istation2 = stathmos3.Osuper,
  supervisor1.Istation3 = stathmos5.Osuper,
  supervisor1.Istation = stathmos1.Osuper,
  DSB2.Istat = stathmos1.Odsb,
  DSB2.Isuperr = supervisor1.Odsb,
  DSB2.Iafter = DSB6.Obefore,
  DSB2.Ibefore = DSB1.Oafter,
  stathmos1.Isuper = supervisor1.Ostation,
  stathmos1.Idsb = DSB2.Ostat,
  stathmos1.Ibefore2 = stathmos2.Onext2,
  stathmos1.Ibefore = stathmos2.Onext;
sync
```

<SyncTrain1 , DSB2.joinedtrain , stathmos1.trains>,

<SyncTrain2 , DSB1.joinedtrain , stathmos2.trains>,

<SyncTrain3 , DSB4.joinedtrain , stathmos3.trains>,

<SyncTrain4 , DSB6.joinedtrain , stathmos5.trains>,

<SyncLeave1 , DSB2.joinedleave , DSB6.joinedtrain , stathmos1.leaving , stathmos5.trains>,

<SyncLeave2 , DSB1.joinedleave , DSB2.joinedtrain , stathmos1.trains , stathmos2.leaving>,

<SyncLeave3 , DSB1.joinedtrain , DSB4.joinedleave , stathmos2.trains , stathmos3.leaving>,

<SyncLeave4 , DSB4.joinedtrain , DSB6.joinedleave , stathmos3.trains , stathmos5.leaving>,

<SyncTrains1 , DSB2.joinedtrain2 , stathmos1.trains2>,

<syncTrains2 , DSB1.joinedtrain2 , stathmos2.trains2>,

<SyncTrains3 , DSB4.joinedtrain2 , stathmos3.trains2>,

<SyncTrains4 , DSB6.joinedtrain2 , stathmos5.trains2>;

edon