ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη εργαλείων για γλώσσες ειδικού σκοπού και εφαρμογή στο διάχυτο υπολογισμό

Ευάγγελος Ε. Νομικός

Επιβλέπων: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΝΟΕΜΒΡΙΟΣ 2009

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη εργαλείων για γλώσσες ειδικού σκοπού και εφαρμογή στο διάχυτο υπολογισμό

Ευάγγελος Ε. Νομικός

Α.Μ.: Μ917

ΕΠΙΒΛΕΠΩΝ:
Ευστάθιος Χατζηευθυμιάδης , Επίκουρος Καθηγητής ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:
Ευστάθιος Χατζηευθυμιάδης , Επίκουρος Καθηγητής ΕΚΠΑ
Ιωάννης Κοτρώνης, Επίκουρος Καθηγητής ΕΚΠΑ

ΝΟΕΜΒΡΙΟΣ 2009

ΠΕΡΙΛΗΨΗ

Στα πρώιμα στάδια της επιστήμης των υπολογιστών, η διαδικασία ανάπτυξης γλωσσών ήταν μια πολύ κοινή δραστηριότητα με μεγάλο ερευνητικό ενδιαφέρον. Πολύ συχνά οι προγραμματιστές σε μια προσπάθεια να αυξήσουν την παραγωγικότητα τους και να αποκρύψουν την πολυπλοκότητα του συστήματος που αναπτύσσουν αυξάνουν το επίπεδο της αφαίρεσης δημιουργώντας γλώσσες ειδικού σκοπού. Σε αντίθεση με τις γλώσσες γενικού σκοπού, οι γλώσσες αυτές στοχεύουν σε ένα πολύ περιορισμένο πεδίο εφαρμογής, με πολύ συγκεκριμένα χαρακτηριστικά. Ωστόσο, ένα βασικό πρόβλημα στην παραπάνω προσέγγιση παραμένει η κατασκευή ενός συνόλου από εργαλεία που θα υποστηρίζουν τη γλώσσα αυτή. Στα πλαίσια της διπλωματικής εργασίας, μελετώνται τα βασικά χαρακτηριστικά της διαδικασίας που βασίζεται σε μοντέλα και η οποία υπόσχεται την αντιμετώπιση του παραπάνω προβλήματος. Επίσης, επιχειρείται η ανάπτυξη μιας νέας γλώσσας ειδικού σκοπού και ο σχεδιασμός και υλοποίηση ενός επεκτάσιμου περιβάλλοντος δημιουργίας και ελέγχου εφαρμογών για πλατφόρμες διάχυτου υπολογισμού, όπου η υποστήριξη από τέτοια περιβάλλοντα έχει αποδειχθεί κρίσιμος παράγοντας για την υιοθέτηση των λύσεων που προτείνονται. Το περιβάλλον αυτό χρησιμοποιεί ως βάση την πλατφόρμα του Eclipse και επεκτείνει τη λειτουργικότητα της χρησιμοποιώντας το μηχανισμό των αρθρωμάτων (plug-ins).

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: ανάπτυξη βασισμένη σε μοντέλα, διάχυτος υπολογισμός

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: γλώσσες ειδικού σκοπού, ανάπτυξη βασισμένη σε μοντέλα, μετα-μοντέλο, μοντέλο, γεννήτορας παραγωγής κώδικα, ολοκληρωμένο περιβάλλον ανάπτυξης

ABSTRACT

In early stages of computer science, the process of language development was a very common activity with a high research interest. Very often, in an effort to increase their productivity and hide the complexity of the system they develop, the programmers increase the abstraction level by creating domain-specific languages. In contrast with the general purpose languages, these languages aim in a very limited application domain with very specific characteristics. However, a basic problem in the above approach remains the development of a toolset that will support this language. In this master thesis, we study the basic characteristics of the process that is based on models and which promise the confrontation of the above problem. Also, the development of a domain-specific language and the design and implementation of an extensible application creation and testing environment for pervasive computing platforms is attempted, where the support of such environments has been proved a critical factor for the adoption of the proposed solutions. This environment is based on the Eclipse platform and extends its functionality using the plug-in mechanism.

SUBJECT AREA: model driven development, pervasive computing

KEYWORDS: domain specific languages, model driven development, metamodel, model, code generator, IDE

ΕΥΧΑΡΙΣΤΙΕΣ

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της εργασίας, κ. Ευστάθιο Χατζηευθυμιάδη, επίκουρο καθηγητή του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού Καποδιστριακού Πανεπιστημίου Αθηνών για την ευκαιρία που μου έδωσε να ασχοληθώ με το ενδιαφέρον αυτό θέμα στα πλαίσια των δραστηριοτήτων της ερευνητικής του ομάδας. Οι χρήσιμες παρατηρήσεις του, η διακριτικότητα του και η εμπιστοσύνη προς τους συνεργάτες του, αναμφισβήτητα είναι στοιχεία του χαρακτήρα του που τον διακρίνουν ως άνθρωπο και ως καθηγητή. Ωστόσο, το ευχάριστο και δημιουργικό κλίμα που επικρατεί στα πλαίσια της ερευνητικής του ομάδας είναι η καλύτερη απόδειξη για όλα τα παραπάνω.

Ένα ακόμα ταξίδι λοιπόν έφτασε στο τέλος του! Η διπλωματική αυτή εργασία, ολοκληρώθηκε! Συνταξιδιώτες, τα μέλη της ομάδας διάχυτου υπολογισμού! Θα ήταν παράλειψη λοιπόν να μην ευχαριστήσω το Βασίλη, το Βασίλη (vpap) και τον Κώστα, υποψήφιους διδάκτορες του τμήματος Πληροφορικής και Τηλεπικοινωνιών, για τις παρατηρήσεις τους καθ' όλη τη διάρκεια εκπόνησης της εργασίας αυτής αλλά και για την ανοχή τους στην επιμονή μου στη λεπτομέρεια στη φάση της υλοποίησης. Ιδιαίτερα, οφείλω ένα μεγάλο ευχαριστώ, στο συνεργάτη και φίλο Οδυσσέα, που με τον τρόπο του, φρόντισε οι ώρες που περάσαμε στο ίδιο γραφείο να είναι ευχάριστες και πολύ δημιουργικές. Έτσι, μπορεί τελικά να μη φτάσαμε στην Ιθάκη, αλλά σίγουρα, είμαστε πλέον περισσότερο πολυταξιδεμένοι…

Νοέμβριος 2009,

Ευάγγελος Νομικός

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του Μεταπτυχιακού Προγράμματος Σπουδών, στην κατεύθυνση «Προηγμένα Πληροφοριακά Συστήματα», του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών. Το αντικείμενο της εργασίας είναι η μελέτη και η αποτίμηση των βασικότερων τεχνικών για την ανάπτυξη υποστηρικτικών εργαλείων για γλώσσες ειδικού σκοπού με στόχο την ανάπτυξη ενός ολοκληρωμένου επεκτάσιμου περιβάλλοντος ανάπτυξης και ελέγχου εφαρμογών για περιβάλλοντα διάχυτου υπολογισμού. Εκτός από τις παραδοσιακές τεχνικές ανάπτυξης γλωσσών που βασίζονται στην ισχυρά θεμελιωμένη θεωρία των μεταγλωττιστών, μια εναλλακτική προσέγγιση βασισμένη σε μοντέλα χρησιμοποιείται ολοένα και περισσότερο τα τελευταία χρόνια. Στα πλαίσια της εργασίας, μελετώνται οι βασικές αρχές και τεχνικές της προσέγγισης αυτής και υιοθετούνται για την επίτευξη του παραπάνω στόχου.

Η ερευνητική ομάδα διάχυτου υπολογισμού του τμήματος Πληροφορικής και Τηλεπικοινωνιών στα πλαίσια των ερευνητικών της δραστηριοτήτων αναπτύσσει μια πλατφόρμα για συνεργατικές (collaborative) εφαρμογές με επίγνωση του περιβάλλοντος (context aware) για κινητούς κόμβους. Το κύριο μέρος της πλατφόρμας αυτής αποτελείται από το κατάλληλο ενδιάμεσο λογισμικό μέσα στο οποίο γίνεται η εγκατάσταση των εφαρμογών. Στα πλαίσια της διπλωματικής, επιχειρείται ο σχεδιασμός και η υλοποίηση μιας γλώσσας μοντελοποίησης εδικού σκοπού και ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης τέτοιων εφαρμογών που σκοπό έχουν να αυξήσουν την διεισδυτικότητα της πλατφόρμας στους τελικούς χρήστες. Ας σημειωθεί ότι, το κείμενο που ακολουθεί συνοδεύεται από πηγαίο κώδικα ο οποίος βρίσκεται στον αντίστοιχο ψηφιακό δίσκο.

ΜΕΡΟΣ Α: ΘΕΜΕΛΙΩΔΕΙΣ ΑΡΧΕΣ – ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

1. ΕΙΣΑΓΩΓΗ

1.1 Γενικά

Στα πρώιμα στάδια της επιστήμης των υπολογιστών, η διαδικασία ανάπτυξης νέων γλωσσών ήταν μια πάρα πολύ κοινή δραστηριότητα με μεγάλο ερευνητικό ενδιαφέρον [19]. Παρόλα αυτά, προς το τέλος της προηγούμενης χιλιετίας, η τέχνη της ανάπτυξης γλωσσών προοριζόταν μόνο για κάποιους πολύ ειδικούς. Ωστόσο, σε όλη την ιστορία του πεδίου της ανάπτυξης λογισμικού, οι προγραμματιστές διαρκώς αναζητούν διάφορους τρόπους να βελτιώσουν την παραγωγικότητα τους, αυξάνοντας το επίπεδο της αφαίρεσης στο οποίο δουλεύουν. Βασικές αρχές του προγραμματισμού όπως το DRY (Don't Repeat Yourself) και ο διαχωρισμός σε υπομονάδες (modularization) στοχεύουν στην υψηλή συνοχή και χαμηλή εξάρτηση μεταξύ των συστατικών του αναπτυσσόμενου λογισμικού, συνεισφέροντας περισσότερο προς την κατεύθυνση αυτή. Παρόλα αυτά, οι εξελίξεις στις παραδοσιακές γλώσσες προγραμματισμού και γλώσσες μοντελοποίησης συνεισφέρουν σχετικά λίγο στην παραγωγικότητα – τουλάχιστον αν τις συγκρίνουμε με την αύξηση της παραγωγικότητας που κερδήθηκε όταν μετακινηθήκαμε από τους assemblers στις γλώσσες τρίτης γενιάς (3GLs) δεκαετίες πριν.

Έτσι, την τελευταία δεκαετία παρατηρείται μια αναγέννηση του ενδιαφέροντος για σχεδιασμό γλωσσών ειδικού σκοπού (domain specific languages) που χρησιμοποιούνται κάπου στη διαδικασία ανάπτυξης λογισμικού. Γλώσσες και εργαλεία που κατασκευάζονται για να λύνουν ένα συγκεκριμένο πρόβλημα τα καταφέρνουν σχεδόν πάντα καλύτερα από τις αντίστοιχες γλώσσες και εργαλεία γενικού σκοπού. Ωστόσο, η κατασκευή γλωσσών για πολύ συγκεκριμένα πεδία εξακολουθεί να αποτελεί ένα πάρα πολύ δύσκολο εγχείρημα με μεγάλες προκλήσεις. Ο σχεδιαστής της γλώσσας (language designer) εκτός από τις εξειδικευμένες γνώσεις στην κατασκευή γλωσσών που πρέπει να διαθέτει, πρέπει να γνωρίζει καλά και τις έννοιες του πεδίου στο οποίο απευθύνεται. Αν λάβει κανείς υπόψη και το γεγονός ότι πολλές φορές είναι αναγκαίο ο τελικός χρήστης (end user) ή ο ειδικός του πεδίου (domain expert), που τις περισσότερες φορές δεν είναι έμπειρος προγραμματιστής, ότι πρέπει να παράξει αποδοτικά και αξιόπιστα εφαρμογές τότε το όλο εγχείρημα γίνεται ακόμα πιο δύσκολο. Για το σκοπό αυτό η ανάπτυξη των εφαρμογών θα πρέπει να πραγματοποιείται σε όσο το δυνατόν υψηλότερο επίπεδο και να γίνεται εκτενής χρήση των εννοιών του πεδίου.

Αν και οι παραδοσιακές τεχνικές ανάπτυξης γλωσσών βασίζονται πάνω στην καλά θεμελιωμένη θεωρία των μεταγλωττιστών (compilers) [8] και χρησιμοποιούνται σχεδόν εξ' ολοκλήρου για την ανάπτυξη γλωσσών υπό μορφή κειμένου, φαίνεται να μη καλύπτουν αποτελεσματικά την προσέγγιση αυτή. Η παραγωγή υποστηρικτικών εργαλείων παραμένει μια χειροκίνητη διαδικασία και αποτελεί ένα χρονοβόρο και περίπλοκο εγχείρημα. Στο σημείο αυτό ένα νέο προγραμματιστικό παράδειγμα για την κατασκευή γλωσσών ειδικού σκοπού έρχεται να κάνει τη διαφορά. Η ανάπτυξη λογισμικού βασισμένη σε μοντέλα (Model Driven Software Development - MDSD) [6] και ειδικότερα η μοντελοποίηση συγκεκριμένου πεδίου (Domain Specific Modeling - DSM) [1] στοχεύει στην αύξηση του επιπέδου αφαίρεσης ανάπτυξης εφαρμογών για συγκεκριμένα πεδία ορίζοντας τη λύση με έννοιες που σχετίζονται άμεσα με το πρόβλημα του πεδίου. Η προσέγγιση αυτή, προσφέρει επίσης τη δυνατότητα ανάπτυξης διαφορετικών διακριτών συντάξεων για τα στιγμιότυπα της γλώσσας πέραν από τη κλασσική μορφή κειμένου. Το κυριότερο πλεονέκτημα όμως της προσέγγισης αυτής, δεν είναι κάτι άλλο πέρα από την αυξανόμενη διάθεση ολοκληρωμένων περιβαλλόντων που υποστηρίζουν τη δημιουργία τέτοιων λύσεων και αυτοματοποιούν σε μεγάλο βαθμό την ανάπτυξη υποστηρικτικών εργαλείων για τις υπό ανάπτυξη γλώσσες. Έτσι, ο σχεδιαστής της γλώσσας μπορεί γρήγορα και με μικρή προσπάθεια να δημιουργήσει όχι μόνο μια νέα γλώσσα αλλά και όλο το αναγκαίο σύνολο από εργαλεία για χρήση της γλώσσας αυτής.

1.2 Αντικείμενο της εργασίας

Σκοπό της εργασίας αποτελεί η μελέτη των τεχνικών ανάπτυξης γλωσσών ειδικού σκοπού καθώς και η διαδικασία παραγωγής ενός συνόλου από εργαλεία που την υποστηρίζουν. Οι περισσότερες νέες ιδέες στην ανάπτυξη λογισμικού συνήθως είναι στην ουσία πραγματικές παραλλαγές πάνω σε παλιές ιδέες. Η διπλωματική αυτή, περιγράφει μια από αυτές, την ολοένα και περισσότερο αναπτυσσόμενη ιδέα μιας κλάσης από περιβάλλοντα και τεχνικές υπό τον όρο ανάπτυξη λογισμικού βασισμένη σε μοντέλα. Η συνεισφορά της εργασίας έγκειται στο γεγονός της χρήσης ενός state-of-the-art περιβάλλοντος ανάπτυξης τέτοιων λύσεων που βασίζεται στην πλατφόρμα του Eclipse, για την παραγωγή μιας νέας γλώσσας ειδικού σκοπού και ενός επεκτάσιμου περιβάλλοντος ανάπτυξης και ελέγχου εφαρμογών που προορίζονται για πλατφόρμες διάχυτου υπολογισμού. Το περιβάλλον αυτό σκοπεύει στη διευκόλυνση της ανάπτυξης έξυπνων εφαρμογών για κινητούς κόμβους ενώ η χρήση της νέας αυτής γλώσσας

αυξάνει το επίπεδο της αφαίρεσης στο οποίο αυτές δημιουργούνται. Με τον τρόπο αυτό, μελετάται ο βαθμός υιοθέτησης και η αύξηση της διεισδυτικότητας μιας πλατφόρμας διάχυτου υπολογισμού από τους τελικούς χρήστες.

Για το σκοπό αυτό, η εργασία διαχωρίζεται σε δύο βασικά μέρη. Στο πρώτο μέρος μελετώνται τα βασικά χαρακτηριστικά των γλωσσών ειδικού σκοπού, παρουσιάζονται οι θεμελιώδεις αρχές της παραδοσιακής ανάπτυξης γλωσσών και της ανάπτυξης βασισμένης σε μοντέλα. Στο δεύτερο μέρος περιγράφεται αναλυτικά ο σχεδιασμός και η υλοποίηση του επεκτάσιμου ολοκληρωμένου περιβάλλοντος ανάπτυξης εφαρμογών και αναλύονται τα βασικά χαρακτηριστικά κάθε επί μέρους συστατικού του.

2. ΠΑΡΑΔΟΣΙΑΚΗ ΑΝΑΠΤΥΞΗ ΓΙΑ ΓΛΩΣΣΕΣ ΕΙΔΙΚΟΥ ΣΚΟΠΟΥ

Η ανάπτυξη μιας γλώσσας ειδικού σκοπού είναι μια δύσκολη διαδικασία, η οποία απαιτεί γνώση του πεδίου (domain) και αρκετή εμπειρία και γνώσεις γύρω από την διαδικασία ανάπτυξης γλωσσών. Σχετικά λίγοι άνθρωποι διαθέτουν και τα δύο παραπάνω χαρακτηριστικά με αποτέλεσμα η απόφαση ανάπτυξης μιας γλώσσας ειδικού σκοπού συχνά να αναβάλλεται απ' αορίστου και οι περισσότερες προσπάθειες συχνά να μην ξεπερνούν το επίπεδο μιας βιβλιοθήκης εφαρμογής (application library).

Στο κεφάλαιο αυτό περιγράφονται βασικές έννοιες γύρω από τη μηχανική γλωσσών ειδικού σκοπού (domain specific language engineering) και παρουσιάζεται ο παραδοσιακός τρόπος με τον οποίο αναπτύσσονται τέτοιου είδους γλώσσες και υποστηρικτικά εργαλεία.

2.1 Γλώσσες ειδικού σκοπού

Οι γλώσσες ειδικού σκοπού (Domain-Specific Languages – DSLs) είναι γλώσσες οι οποίες κατασκευάζονται για ένα συγκεκριμένο πεδίο εφαρμογής και είναι σχεδιασμένες ώστε να είναι χρήσιμες για ένα συγκεκριμένο σύνολο από ενέργειες. Σε αντίθεση με τις γλώσσες γενικού σκοπού (General Purpose Languages - GPLs) κάποιος θα μπορούσε να περιγράψει μια DSL ως μια συγκεκριμένη διάλεκτο πιθανά μη χρήσιμη σε διαφορετικά πεδία. Παραδείγματα τέτοιων γλωσσών υπάρχουν πάρα πολλά με ορισμένα πολύ συχνά χρησιμοποιούμενα να παρουσιάζονται στον πίνακα 1.

Πίνακας 1: Παραδείγματα γλωσσών ειδικού σκοπού.

DSL	Χρήση
SQL	γλώσσα επερωτήσεων για σχεσιακές βάσεις
Ant, Make	γλώσσες για την παραγωγή συστημάτων λογισμικού
Yacc, Bison, ANTLR	γλώσσες παραγωγής parser
CSS	γλώσσα περιγραφής φύλλων στυλ
HTML	γλώσσα σήμανσης υπερκειμένου
VHDL	γλώσσα περιγραφής υλικού (hardware)

Η πιο κοινή σύνταξη που υιοθετούν οι γλώσσες αυτές είναι υπό μορφή κειμένου (textual) ενώ σε περιπτώσεις που παρέχεται η δυνατότητα γραφικής σύνταξης (graphical syntax) οι γλώσσες αυτές περιγράφονται και από τον όρο γραφικές γλώσσες ειδικού σκοπού (Domain-Specific Visual Languages - DSVLs). Στην βιβλιογραφία γενικότερα, οι γλώσσες αυτές συχνά εμφανίζονται και με τους όρους application-oriented, special purpose, specialized, task-specific, ή application languages.

2.1.1 Χαρακτηριστικά γλωσσών ειδικού σκοπού

Μια γλώσσα ειδικού σκοπού παρουσιάζει κάποια ιδιαίτερα χαρακτηριστικά τα οποία τις περισσότερες φορές την διακρίνουν από μια γλώσσα γενικού σκοπού. Το βασικότερο ίσως χαρακτηριστικό είναι το πολύ καλά καθορισμένο και περιορισμένο πεδίο εφαρμογής. Η γλώσσα προορίζεται για την ανάπτυξη εφαρμογών με πολύ συγκεκριμένα χαρακτηριστικά και δυνατότητες που θα υιοθετηθούν από ένα πολύ αυστηρά καθορισμένο περιβάλλον. Για το σκοπό αυτό, η σύνταξη της γλώσσας χρησιμοποιεί τις κατάλληλες έννοιες και δομές του συγκεκριμένου πεδίου που εύκολα γίνονται κατανοητές από τους ειδικούς του πεδίου αυτού. Με τον τρόπο αυτό, εξασφαλίζεται μεγάλο επίπεδο εκφραστικότητας και ευκολίας στη χρήση συγκρινόμενο πάντα με το αντίστοιχο επίπεδο που επιτυγχάνεται με τη χρήση μιας γλώσσας γενικού σκοπού για την ανάπτυξη εφαρμογών στο πεδίο αυτό.

Επίσης, τις περισσότερες φορές ο αριθμός των χρηστών για τους οποίους προορίζεται μια DSL είναι αρκετά περιορισμένος εξαιτίας του περιορισμένου πεδίου εφαρμογής και εξειδίκευσης που απαιτείται. Μια DSL συνήθως χρησιμοποιείται από χρήστες με μεγάλη εξοικείωση με τις έννοιες του πεδίου χωρίς όμως αυτό να αποτελεί γενικό χαρακτηριστικό καθώς μπορούν να υπάρξουν DSLs οι οποίες χρησιμοποιούνται από μια ευρεία ομάδα χρηστών. Για παράδειγμα, η γλώσσα μακροεντολών του Microsoft Excel η οποία του προσθέτει προγραμματιστικές δυνατότητες είναι μια τέτοια γλώσσα ειδικού σκοπού. Αντίθετα γλώσσες, οι οποίες αναπτύσσονται στα πλαίσια κάποιων έργων (projects) ώστε να διευκολύνουν κάποιες συγκεκριμένες διαδικασίες χρησιμοποιούνται από έναν αρκετά περιορισμένο αριθμό χρηστών και μάλιστα για μικρά χρονικά διαστήματα καθώς μετά το πέρας του έργου οι γλώσσες αυτές παύουν να χρησιμοποιούνται. Επιπλέον, ας σημειωθεί ότι σε αντίθεση τις με τις γλώσσες γενικού σκοπού τις περισσότερες φορές για τις DSLs δεν υπάρχει η απαίτηση να είναι εκτελέσιμες.

2.1.2 Βασικοί τύποι γλωσσών ειδικού σκοπού

Σύμφωνα με το [34], μπορούμε να διακρίνουμε δυο βασικούς τύπους γλωσσών ειδικού σκοπού: τις εξωτερικές (external) και τις εσωτερικές (internal). Μια εξωτερική γλώσσα ειδικού σκοπού είναι ουσιαστικά μια νέα γλώσσα η οποία δε βασίζεται σε κάποια προϋπάρχουσα γλώσσα βάσης. Η γλώσσα αυτή αναπτύσσεται από την αρχή και ο σχεδιαστής της γλώσσας μπορεί να χρησιμοποιήσει οποιαδήποτε σύνταξη και σημασιολογία επιθυμεί. Ως αποτέλεσμα, η γλώσσα μπορεί να περιέχει δομές και έννοιες που αντιστοιχούν σε αντίστοιχες έννοιες του πεδίου και είναι εύκολα κατανοητές από τους ειδικούς του πεδίου αυτού. Για τον ορισμό της γλώσσας, ο σχεδιαστής χρησιμοποιεί κάποια δημοφιλή τεχνική περιγραφής γλωσσών και η σύνταξη της γλώσσας στην οποία καταλήγει περιορίζεται μόνο από την ικανότητα του να κατασκευάσει τα αντίστοιχα εργαλεία που θα μπορούν να διαβάζουν/αναλύουν τα στιγμιότυπα της γλώσσας και να παράγουν το επιθυμητό αποτέλεσμα. Για το σκοπό αυτό, χρησιμοποιούνται parser generator και compiler compiler εργαλεία [35, 36, 37, 38] που μπορούν να βοηθήσουν στην κατασκευή αρκετά σύνθετων γλωσσών. Συνήθως οι περισσότερες από τις εξωτερικές γλώσσες δεν είναι εκτελέσιμες γλώσσες και γι’ αυτό απαιτείται η μετατροπή των δομών που αυτές υποστηρίζουν σε δομές μιας άλλης εκτελέσιμης γλώσσας βάσης. Για παράδειγμα μια εξωτερική γλώσσα ειδικού σκοπού μπορεί να μετατρέπεται σε κώδικα Java ο οποίος εν συνεχεία εκτελείται πάνω από κάποια πλατφόρμα στόχο. Έννοιες και τεχνικές από την θεωρία των μεταγλωττιστών είναι βασικές για το σχεδιασμό μιας τέτοιας γλώσσας.

Παρόλα αυτά, μια κοινά συχνή αντίρρηση για τις εξωτερικές γλώσσες ειδικού σκοπού είναι το πρόβλημα της κακοφωνίας της γλώσσας (language cacophony problem). Αυτό σχετίζεται με το πρόβλημα ότι καινούριες γλώσσες μαθαίνονται σχετικά δύσκολα και απαιτούν πρώτα κάποιου είδους εξοικείωση από τους χρήστες. Τις περισσότερες φορές όμως οι γλώσσες ειδικού σκοπού είναι πολύ απλές και περιορισμένες γεγονός που τις καθιστά εύκολες στην εκμάθηση κάτι που ενισχύεται και από την κλειστότητα του πεδίου.

Αντίθετα μια εσωτερική γλώσσα ειδικού σκοπού μορφοποιεί συνήθως κάποια προϋπάρχουσα γλώσσα βάσης. Μια τέτοια γλώσσα είτε επεκτείνει τη γλώσσα βάσης προσθέτοντας νέες δομές και έννοιες, είτε την περιορίζει χρησιμοποιώντας μόνο ένα υποσύνολο από τις βασικές έννοιες και δομές αυτής. Ως αποτέλεσμα η σύνταξη της περιορίζεται σε μεγάλο βαθμό από τη σύνταξη της γλώσσας βάσης και η ισχύς της εξαρτάται από την ισχύ της γλώσσας αυτής. Παρόλα αυτά η γλώσσα βάσης μπορεί να διαθέτει πολλά εργαλεία ανάπτυξης τα οποία με τις κατάλληλες τροποποιήσεις να

μπορούν να υποστηρίξουν και τη γλώσσα ειδικού σκοπού. Τις περισσότερες φορές όμως, οι τροποποιήσεις που απαιτούνται δεν είναι πάντα εύκολες και υλοποιήσιμες σε εύλογα χρονικά διαστήματα με αποτέλεσμα τα εργαλεία αυτά να μη είναι εφικτό να χρησιμοποιηθούν άμεσα.

2.2 Στάδια ανάπτυξης μιας γλώσσας ειδικού σκοπού

Όπως και οποιοδήποτε άλλο έργο λογισμικού έτσι και η διαδικασία ανάπτυξης μια γλώσσας ειδικού σκοπού αποτελείται από τα ίδια βασικά στάδια ανάπτυξης: απόφαση (decision), ανάλυση (analysis), σχεδιασμός (design), υλοποίηση (implementation) και εγκατάσταση στο πεδίο εφαρμογής [14]. Στη πράξη όμως, μιας DSL δεν είναι μια ακολουθιακή διεργασία. Η διεργασία απόφασης μπορεί να επηρεάζεται από πρώιμη ανάλυση, η οποία με τη σειρά της μπορεί να παρέχει απαντήσεις από απρόβλεπτα ερωτηματικά που προκύπτουν κατά τη διάρκεια του σχεδιασμού, και ο σχεδιασμός συχνά επηρεάζεται από θέματα υλοποίησης. Πριν την έναρξη της διαδικασίας ανάπτυξης πρέπει να αναλυθεί το όφελος που θα προκύψει από την δημιουργία της DSL καθώς η αρχική της ανάπτυξη, συντήρηση και μελλοντική της εξέλιξη απαιτεί χρόνο και χρήμα. Σε όλα τα στάδια ανάπτυξης, οι σχεδιαστές της γλώσσας, οι ειδικοί του πεδίου και οι τελικοί χρήστες συνεργάζονται ώστε να επιτευχθεί το κατάλληλο επίπεδο αφαίρεσης. Στα αρχικά στάδια, ο σχεδιαστής της γλώσσας σε στενή συνεργασία με τον ειδικό του πεδίου αναλύει και μελετά το πεδίο εφαρμογής ώστε να εντοπίσει τις βασικές έννοιες και τις μεταξύ τους συσχετίσεις. Σκοπός είναι μόνο η αντιμετώπιση της πολυπλοκότητας του πεδίου ενώ οι λεπτομέρειες υλοποίησης και τυχόν άλλες όχι βασικές πολυπλοκότητες δεν εξετάζονται στη φάση αυτή.

Οι DSLs χρησιμοποιούνται από τους ειδικούς του πεδίου που τις περισσότερες φορές δεν είναι προγραμματιστές. Στην πράξη υπάρχουν πολλά παραδείγματα σήμερα όπου μη-προγραμματιστές χρησιμοποιούν εύκολα και με αποδοτικό τρόπο DSLs, όταν αυτές παρέχουν το κατάλληλο επίπεδο αφαίρεσης. Έτσι είναι πολύ σημαντικό οι DSLs να είναι διαισθητικές στους χρήστες (user) και να προσφέρουν το σωστό επίπεδο συντακτικής και σημασιολογικής αφαίρεσης. Από την στιγμή που οι έννοιες του πεδίου έχουν γίνει σαφώς κατανοητές από τον σχεδιαστή της γλώσσας επόμενο στάδιο είναι το στάδιο υλοποίησης όπου με χρήση των κατάλληλων τεχνικών και εργαλείων υλοποιείται η γλώσσα. Η υιοθέτηση του παραγόμενου αποτελέσματος θα γίνει από τους τελικούς χρήστες και τους ειδικούς του πεδίου έπειτα από την εγκατάσταση της λύσης σε πραγματικές συνθήκες στο πεδίο εφαρμογής.

2.3 Περιγραφές γλωσσών

Στον τομέα της ανάπτυξης γλωσσών προγραμματισμού υπάρχουν αρκετοί φορμαλισμοί οι οποίοι χρησιμοποιούνται για τον ορισμό μιας γλώσσας. Οι φορμαλισμοί αυτοί ονομάζονται περιγραφές γλωσσών (language descriptions) και μπορούν κάλλιστα να χρησιμοποιηθούν στην ανάπτυξη γλωσσών ειδικού σκοπού.

Μια γλώσσα ειδικού σκοπού μπορεί να είναι ένα αρκετά σύνθετο κατασκεύασμα στο οποίο πρέπει να υπάρχει σαφής διαχωρισμός μεταξύ της σύνταξης (syntax) από την σημασιολογία (semantics). Η σύνταξη αφορά όλες εκείνες τις έγκυρες δομές τις οποίες μπορεί να χρησιμοποιήσει ο χρήστης για τη δημιουργία ενός στιγμιότυπου της γλώσσας ενώ η σημασιολογία αφορά την κατανόηση της πληροφορίας η οποία σύμφωνα με μια ερμηνεία αντιστοιχεί μια πολύ συγκεκριμένη σημασία για κάθε σύμβολο ή δομή που χρησιμοποιείται.

Οποιοσδήποτε έχει μελετήσει μια ξένη γλώσσα γνωρίζει καλά ότι μια γραμματική είναι ένα βιβλίο με κανόνες και παραδείγματα που περιγράφουν και διδάσκουν τη γλώσσα. Έτσι και στον τομέα της ανάπτυξης γλωσσών, ο πιο συχνός τρόπος περιγραφής μιας γλώσσας είναι οι περιγραφές γλωσσών βασισμένες σε γραμματικές. Η χρήση των γραμματικών καθιερώθηκε από την ανάπτυξη των πρώτων γλωσσών προγραμματισμού και χρησιμοποιείται σε μεγάλο βαθμό ακόμα και σήμερα. Ως αποτέλεσμα, υπάρχει μια πλούσια βιβλιογραφία και γνώση πάνω στην διαδικασία ανάπτυξης μιας γλώσσας με χρήση γραμματικής καθώς και αρκετά εργαλεία που την υποστηρίζουν.

Ορισμός: Μια γραμματική είναι μια ειδική κατηγορία περιγραφής γλωσσών που χρησιμοποιείται για την περιγραφή γλωσσών υπό μορφή κειμένου. Αποτελεί ένα σύστημα από κανόνες πάνω από ένα σύνολο από σύμβολα (αλφάβητο) που περιγράφει ποιες πιθανές ακολουθίες συμβόλων αποτελούν έγκυρα στιγμιότυπα της γλώσσας.

Στην επίσημη θεωρία γλωσσών περιγράφονται τέσσερις τύποι γλωσσών υπό μορφή κειμένου με τέσσερις αντίστοιχους τύπους γραμματικών (γνωστή ως ιεραρχία του Chomsky). Οι γραμματικές αυτές με τη σειρά τους ισοδυναμούν με τα αντίστοιχα αυτόματα τα οποία και αυτά διαχωρίζονται σε τέσσερις κατηγορίες. Η πιο σύνθετη κατηγορία γλώσσας η οποία αντιστοιχεί στον τύπο αυτομάτου το οποίο είναι αρκετά απλό ώστε πρακτικά να μπορεί να υλοποιηθεί από τους σημερινούς υπολογιστές είναι η κατηγορία των γραμματικών χωρίς συμφραζόμενα (context-free grammars). Οι

φορμαλισμοί οι οποίοι βασίζονται στις γραμματικές αυτές, χρησιμοποιούνται σχεδόν αποκλειστικά για τον ορισμό γλωσσών υπό μορφή κειμένου.

Ωστόσο, υπάρχουν αρκετά διαφορετικά στυλ συμβολισμού για γραμματικές χωρίς συμφραζόμενα για ορισμό γλωσσών προγραμματισμού, καθεμία με πάρα πολλές παραλλαγές όπου όμως όλες είναι λειτουργικά ισοδύναμες. Τα δύο πιο βασικά στυλ είναι το Backus-Naur Form (BNF) το οποίο αρχικά χρησιμοποιήθηκε για τον ορισμό της ALGOL 60 από τους John Backus και Peter Naur και το Extended Backus-Naur Form (EBNF) [10] [11]. Ειδικότερα, οι BNF γραμματικές αποτελούνται από ένα σύνολο από σύμβολα τα οποία χωρίζονται σε τερματικά (terminals) και μη-τερματικά (non-terminals). Επίσης, υπάρχει ένα σύνολο από κανόνες παραγωγής (production rules) που αντιστοιχούν ένα μη-τερματικό σύμβολο σε μια ακολουθία από σύμβολα. Η ομαδοποίηση συμβόλων μπορεί να δηλωθεί χρησιμοποιώντας παρενθέσεις () ενώ η δομή της επιλογής σε έναν κανόνα παραγωγής συμβολίζεται με το σύμβολο |. Έτσι, ένα δέντρο συντακτικής ανάλυσης (parse tree) μπορεί να δημιουργηθεί ξεκινώντας από ένα αρχικό σύμβολο (start symbol) και εφαρμόζοντας κανόνες μέχρις ότου μόνο τερματικά σύμβολα να έχουν απομείνει.

Εναλλακτικά, οι γραμματικές χωρίς συμφραζόμενα συχνά κατασκευάζονται ώστε να είναι ταυτόχρονα πιο συμπαγής και πιο ευανάγνωστες εισάγοντας διάφορες συντμήσεις για συχνά χρησιμοποιούμενες δομές. Κοινές επεκτάσεις αποτελεί η χρήση των συμβόλων

- [] ή ? , για προαιρετικά στοιχεία
- * , επανάληψη στοιχείου καμία ή περισσότερες φορές
- + , επανάληψη στοιχείου μία ή περισσότερες φορές

Ο παραπάνω συμβολισμός για τις γραμματικές καλείται Extended BNF (EBNF). Οι επεκτάσεις μιας EBNF γραμματικής δεν αυξάνουν την εκφραστική της δύναμη καθώς όλοι οι συνεπαγόμενοι κανόνες μπορούν να κατασκευαστούν από ρητούς και τότε προκύπτει μια γραμματική χωρίς συμφραζόμενα σε BNF συμβολισμό. Ωστόσο, η ισχύς τους βρίσκεται στην φιλικότητα προς τον χρήστη.

Παρόλα αυτά, πολύ συχνά οι σύγχρονες γλώσσες προγραμματισμού χρησιμοποιούν διαφορετικούς συμβολισμούς πέραν της μορφής κειμένου. Οι textual γλώσσες είναι κατά βάση συμβολικές και οι βασικές τους συντακτικές εκφράσεις αποτελούνται από ακολουθίες χαρακτήρων. Αντίθετα, στις διαγραμματικές γλώσσες, ή στους γραφικούς φορμαλισμούς, βασικές εκφράσεις περιλαμβάνουν γραμμές, τόξα, καμπύλες, σχήματα

και μηχανισμούς σύνθεσης που περικλείουν έννοιες συνδεσιμότητας (connectivity), διαίρεσης (partitioning) και εγκλεισμού (insideness). Έτσι υπάρχει η απαίτηση για περιγραφές γλωσσών οι οποίες θα μπορούν να ορίσουν τέτοιου είδους γλώσσες. Για τις γλώσσες υπό μορφή κειμένου, η πιο διαδεδομένη μορφή ορισμού είναι οι γραμματικές, ενώ για τις γραφικές γλώσσες υπάρχουν δύο κύριες προσεγγίσεις. Η μια περιλαμβάνει τις γραφικές γραμματικές (graph grammars) [28] οι οποίες επεκτείνουν έννοιες γραμματικών από γλώσσες υπό μορφή κειμένου σε διαγράμματα. Η δεύτερη προσέγγιση περιλαμβάνει ένα είδος διαγράμματος οντοτήτων συσχετίσεων – συγκεκριμένα UML διαγράμματα κλάσεων – για την μοντελοποίηση της αφηρημένης σύνταξης μιας διαγραμματικής γλώσσας. Αν και τα διαγράμματα κλάσεων φαίνεται να είναι πιο διαισθητικά από τις γραφικές γραμματικές, είναι επίσης λιγότερο εκφραστικά.

Στο σημείο αυτό ας σημειωθεί, ότι όλες οι παραπάνω περιγραφές γλωσσών αποτελούν παραδείγματα γλωσσών ειδικού σκοπού. Για παράδειγμα ο BNF φορμαλισμός για την περιγραφή της σύνταξης είναι ένα παράδειγμα DSL με ένα φτωχό δηλωτικό χαρακτήρα που απλά ενεργεί ως μια γλώσσα εισόδου για ένα γεννήτορα parser.

2.4 Κατηγορίες υποστηρικτικών εργαλείων χρήστη

Εκτός από τον σχεδιασμό μιας γλώσσας σημαντικό χαρακτηριστικό διείσδυσης και υιοθέτησης από τους τελικούς χρήστες παίζει το σύνολο των διαθέσιμων εργαλείων (tool set) που θα υποστηρίζουν την ανάπτυξη με τη γλώσσα αυτή. Τα εργαλεία αυτά πρέπει να υποστηρίζουν τόσο την ανάπτυξη όσο και τον έλεγχο των εφαρμογών που αναπτύσσονται. Τα πιο συνήθη εργαλεία που μπορεί να συναντήσει κανείς είναι:

- οι επεξεργαστές (editors): εργαλεία που επιτρέπουν στο χρήστη να δημιουργήσει ένα στιγμιότυπο της γλώσσας από την αρχή ή να τροποποιήσει ένα υπάρχον. Ανάλογα με τη μορφή της γλώσσας μπορεί να υπάρχουν επεξεργαστές για επεξεργασία κειμένου ή γραφικοί επεξεργαστές.
- οι μετατροπείς (transformers): εργαλεία που μετατρέπουν ένα στιγμιότυπο της γλώσσας σε κάποια άλλη μορφή. Πολύ γνωστά παραδείγματα τέτοιων εργαλείων είναι οι μεταγλωττιστές, οι μετατροπείς μοντέλων (model transformers), εργαλεία αντίστροφης μηχανικής (reverse engineering tools), γεννήτορες τεκμηρίωσης (documentation generators) και γεννήτορες κώδικα (code generators).

- οι executors: εργαλεία τα οποία εκτελούν ένα στιγμιότυπο της γλώσσας. Πολύ γνωστά παραδείγματα είναι οι αποσφαλματωτές (debuggers), οι προσομοιωτές (simulators), οι εικονικές μηχανές (virtual machines), και τα αυτοματοποιημένα εργαλεία ελέγχου (automated testers).
- οι αναλυτές (analyzers): εργαλεία που αναλύουν ένα στιγμιότυπο γλώσσας και παράγουν κάποια κρίση/αναφορά για αυτό. Δημοφιλή παραδείγματα είναι εκτιμητές κόστους (cost estimators) – π.χ. βασισμένοι στο πλήθος των γραμμών του πηγαίου κώδικα – επικυρωτές μοντέλων (model verifiers) – π.χ για έλεγχο αποφυγής από αδιέξοδα.

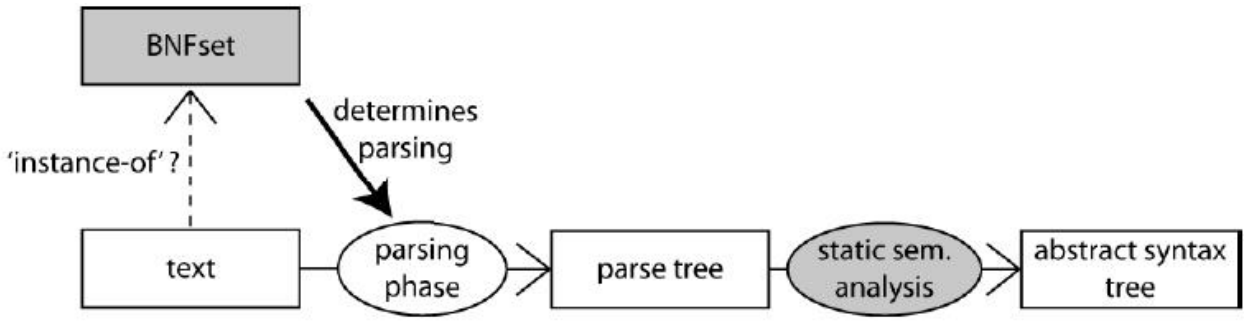
Επιπλέον, οι χρήστες των γλωσσών χρησιμοποιούν εργαλεία που υποστηρίζουν την όλη διαδικασία της ανάπτυξης μια εφαρμογής: για παράδειγμα, εργαλεία διαχείρισης εκδόσεων (version control tools) και αναφοράς σφαλμάτων (bug reporting). Συνήθως αυτά τα εργαλεία είναι εφαρμόσιμα γενικά και όχι συγκεκριμένα για μια γλώσσα.

Ωστόσο, τις περισσότερες φορές, για το χρήστη της γλώσσας, η διαχωριστική γραμμή μεταξύ όλων αυτών των εργαλείων είναι κάπως θολή. Δεν είναι λίγες οι φορές όπου όλα αυτά τα εργαλεία ενσωματώνονται σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE) και προσφέρονται ως σύνολο. Για παράδειγμα, όταν ένας χρήστης θέλει να εκτελέσει ένα πρόγραμμα γραμμένο στη γλώσσα Java στο ολοκληρωμένο περιβάλλον του Eclipse [32], το περιβάλλον εγγυάται ότι οποιεσδήποτε αλλαγές στον επεξεργαστή (editor) αποθηκεύονται, ότι ο μεταγλωττιστής τρέχει και ότι bytecode παράγεται, και ότι αυτό το bytecode εκτελείται από μία εικονική μηχανή. Για το χρήστη, ότι φαίνεται να είναι ένα και μόνο task περιλαμβάνει έναν αριθμό από εργαλεία.

2.5 Ανάπτυξη υποστηρικτικών εργαλείων

Για ένα μηχανικό μιας γλώσσας (language engineer) μια πάρα πολύ σημαντική αποστολή είναι ο λεπτομερής σχεδιασμός και ορισμός της γλώσσας ενώ για την υιοθέτηση της γλώσσας από τους τελικούς χρήστες ο μηχανικός πρέπει να αναπτύξει και το κατάλληλο σύνολο από εργαλεία που θα υποστηρίζουν αποτελεσματικά τη γλώσσα αυτή. Τις περισσότερες φορές, η ανάπτυξη των εργαλείων αυτών από το μηδέν δεν είναι μια εύκολη διαδικασία και απαιτεί μεγάλη εμπειρία στη χρήση εξειδικευμένων τεχνικών. Το μεγαλύτερο μέρος της ανάπτυξης γίνεται χειροκίνητα και συνεπώς καθιστά το όλο εγχείρημα μια πάρα πολύ χρονοβόρα και κοπιαστική διαδικασία με αναμφίβολα πολλές φορές αποτελέσματα.

Ωστόσο, στην περίπτωση γλωσσών υπό μορφή κειμένου η καλά θεμελιωμένη θεωρία ανάπτυξης μεταγλωττιστών [8] μπορεί να συνεισφέρει εν μέρη με τεχνικές και εργαλεία προς αυτή την κατεύθυνση. Σύμφωνα με τον παραδοσιακό τρόπο ανάπτυξης γλωσσών, όταν μια νέα textual γλώσσα δημιουργείται, η κύρια δραστηριότητα είναι η παραγωγή ενός συνόλου από (E)BNF κανόνες που περιγράφουν την αφηρημένη σύνταξης της. Εν συνεχεία, ο κατάλληλος parser για την επεξεργασία έγκυρων στιγμιότυπων της γλώσσας μπορεί εύκολα να κατασκευαστεί με αυτόματο τρόπο, χρησιμοποιώντας κάποιο από τα διαθέσιμα εργαλεία παραγωγής parser όπως τα Yacc [38], ANTLR [35], JavaCC [36] κ.α. Με την βοήθεια του parser και έχοντας ορίσει την κατάλληλη σημασιολογία για τα στοιχεία της γλώσσας, το εκάστοτε στιγμιότυπο της γλώσσας μετατρέπεται σε ένα αφηρημένο συντακτικό δέντρο, όπου χειροκίνητα πλέον μπορεί να διασχιστεί και να παράξει την κατάλληλη έξοδο (εικόνα 1).



Εικόνα 1: Τα βασικότερα βήματα επεξεργασίας ενός στιγμιότυπου μιας γλώσσας υπό μορφή κειμένου.

Η παραπάνω διαδικασία όπως περιγράφηκε, σε καμία περίπτωση δεν παράγει κάποιο ολοκληρωμένο εργαλείο το οποίο μπορεί να χρησιμοποιηθεί κατευθείαν από τους χρήστες της γλώσσας. Ωστόσο εργαλεία που επιτρέπουν την αυτόματη παραγωγή ενός parser και εξειδικευμένες τεχνικές (π.χ. background parsing) μπορούν να αξιοποιηθούν από το σχεδιαστή της γλώσσας ώστε να αναπτύξει εργαλεία επεξεργασίας (π.χ επεξεργαστής κειμένου) με πολύ προχωρημένες δυνατότητες.

Βέβαια, μελετώντας κανείς τη βιβλιογραφία θα συναντήσει και μια άλλη εναλλακτική προσέγγιση στον τρόπο με τον οποίο επιχειρείται η ανάπτυξη υποστηρικτικών εργαλείων για νέες γλώσσας. Στο επίκεντρο της προσέγγισης αυτής βρίσκεται η έννοια του γεννήτορα εργαλείων (tool generator). Ένας γεννήτορας εργαλείων είναι μια ειδική κατηγορία μετατροπέα ο οποίος παίρνει ως είσοδο την περιγραφή της γλώσσας και παράγει κατευθείαν τον πηγαίο κώδικα για το εργαλείο το οποίο ο χρήστης της γλώσσας πρόκειται να χρησιμοποιήσει. Ας σημειωθεί ότι οι γεννήτορες εργαλείων σε

καμιά περίπτωση δεν είναι κάτι καινούριο. Κάποιες ενδιαφέρουσες προσπάθειες γεννητόρων υπήρχαν και παλαιότερα χωρίς όμως η έρευνα στο πεδίο αυτό να έχει αναπτυχθεί πλήρως. Κάποιες πρόσφατες προσεγγίσεις προς αυτή την κατεύθυνση περιγράφονται και στο κεφάλαιο το οποίο ακολουθεί.

3. ΑΝΑΠΤΥΞΗ ΒΑΣΙΣΜΕΝΗ ΣΕ ΜΟΝΤΕΛΑ

Οι τεχνολογίες πληροφορικής εξαπλώνονται ολοένα και περισσότερο σε όλους τους τομείς της καθημερινής ζωής, οδηγώντας σε ένα αυξανόμενο σύνολο από πληροφορίες και εφαρμογές πολύ υψηλής πολυπλοκότητας. Παραδοσιακές μέθοδοι παραγωγής λογισμικού και διαχείρισης δεδομένων αδυνατούν να αντιμετωπίσουν αυτή την αυξανόμενη πολυπλοκότητα. Νέοι τρόποι αντιμετώπισης της πολυπλοκότητας χρησιμοποιούν υψηλότερα επίπεδα αφαίρεσης και περιγράφουν συστήματα μέσω μοντέλων. Ειδικότερα το OMG (Object Management Group) [44] προωθεί τον όρο αρχιτεκτονική βασισμένη σε μοντέλα (Model Driven Architecture – MDA) με τον οποίο αναφέρεται στην ανάπτυξη λογισμικού μέσω μοντέλων υψηλού επιπέδου.

Ωστόσο, πίσω από τη νέα αυτή τάση της ανάπτυξης λογισμικού βασισμένου σε μοντέλα η κεντρική ιδέα δεν αποτελεί κάτι το πρωτοποριακό. Σε όλη τη διαδικασία ανάπτυξης λογισμικού οι προγραμματιστές χρησιμοποιούν μοντέλα ώστε να αποκρύψουν την πολυπλοκότητα του συστήματος που αναπτύσσουν και να αποφύγουν τον πλεονασμό δουλεύοντας σε υψηλότερα επίπεδα αφαίρεσης. Παρόλα αυτά μια ολοκληρωμένη λύση βασισμένη σε μοντέλα αποτελείται από τρία βασικά χαρακτηριστικά μέσω των οποίων επιχειρείται εκτός από την αύξηση του επιπέδου της αφαίρεσης και η αυτοματοποίηση πολλών διαδικασιών.

3.1 Τα βασικά συστατικά

Κατά την διαδικασία ανάπτυξης βασισμένης σε μοντέλα ένας έμπειρος προγραμματιστής πρέπει να κατασκευάσει τρία πράγματα, τα οποία μαζί θα αποτελέσουν το περιβάλλον ανάπτυξης για άλλους προγραμματιστές σε αυτό το πεδίο. Τα τρία αυτά πράγματα είναι τα ακόλουθα:

- μια γλώσσα μοντελοποίησης ειδική για το πεδίο (domain-specific modeling language)
- ένα γεννήτορα κώδικα συγκεκριμένο για το πεδίο (domain-specific code generator)
- ένα πλαίσιο εκτέλεσης (domain framework)

3.1.1 Η γλώσσα μοντελοποίησης

Η γλώσσα μοντελοποίησης αποτελεί το βασικότερο συστατικό μιας λύσης η οποία βασίζεται πάνω σε μοντέλα. Όπως και στην παραδοσιακή περίπτωση ανάπτυξης μιας γλώσσας ειδικού σκοπού, η γλώσσα ορίζει τα όρια του κόσμου μας και καθορίζει τι μπορούμε να περιγράψουμε καθώς και τι δε μπορούμε. Αποτελεί μια τοπική «διάλεκτο» που κάνει χρήση εννοιών του πεδίου που μοντελοποιείται και είναι πολύ πιθανόν να μην είναι χρήσιμη σε άλλα πεδία προβλήματος.

Ορισμός: Ένα πεδίο είναι μια περιοχή ενδιαφέροντος που σχετίζεται με μια συγκεκριμένη προσπάθεια ανάπτυξης.

Η πιο συνηθισμένη γλώσσα μοντελοποίησης είναι η UML με την κύρια διαφορά ότι η UML και τα αντίστοιχα εργαλεία είναι ένα μεγάλο και σύνθετο κατασκεύασμα. Για την ανάπτυξη μια γλώσσας μοντελοποίησης ειδικού σκοπού πρώτα από όλα πρέπει να οριστεί η αφηρημένη σύνταξη (abstract syntax). Σε αντίθεση με την παραδοσιακή προσέγγιση των γλωσσών υπό τη μορφή κειμένου όπου η γλώσσα ορίζεται από μια (E)BNF γραμματική μια γλώσσα μοντελοποίησης ορίζεται με την βοήθεια ενός μεταμοντέλου (metamodel) [5].

Η χρήση των μεταμοντέλων για την ανάπτυξη γλωσσών, πρώτα αναπτύχθηκε και υιοθετήθηκε από το OMG μέσω του προτύπου MOF [45]. Σύμφωνα όμως με το [29], ένα μεταμοντέλο M ορίζεται ότι αποτελείται από τα εξής χαρακτηριστικά

- ένα σύνολο από ονομαστικές μετακλάσεις (metaclasses) C εκ των οποίων μία ονομάζεται ρίζα (root),
- ένα σύνολο από γενικεύσεις (generalisations) G,
- ένα σύνολο από ονομαστικές απαριθμήσεις (enumerations).

Οι γενικεύσεις σχηματίζουν μια ιεραρχία κλάσεων αντιστοιχίζοντας μια υποκλάση στην υπερκλάση της, δημιουργώντας ένα γράφο χωρίς κύκλους. Κάθε στιγμιότυπο μιας μετακλάσης ονομάζεται στοιχείο του μοντέλο (model element). Κάθε μετακλάση C_i ορίζει ένα σύνολο από ιδιότητες οι οποίες ορίζονται ρητά ή κληρονομούνται μεταβατικά από οποιαδήποτε από τις υπερκλάσεις. Τα ονόματα των ιδιοτήτων πρέπει να είναι μοναδικά και ορίζουν τα λεγόμενα slots των στοιχείων μοντελοποίησης, περιγράφοντας το είδος των συνδέσεων που μπορεί να υπάρχουν ανάμεσα στα στοιχεία μοντελοποίησης. Μια ιδιότητα P αποτελείται από ένα όνομα, το εύρος πληθικότητας και την μετακλάση ή απαρίθμηση στόχο. Το όνομα χρησιμοποιείται για τον διαχωρισμό των ιδιοτήτων, το εύρος πληθικότητας ορίζει πόσες συνδέσεις σε στιγμιότυπα του στόχου το slot στόχου

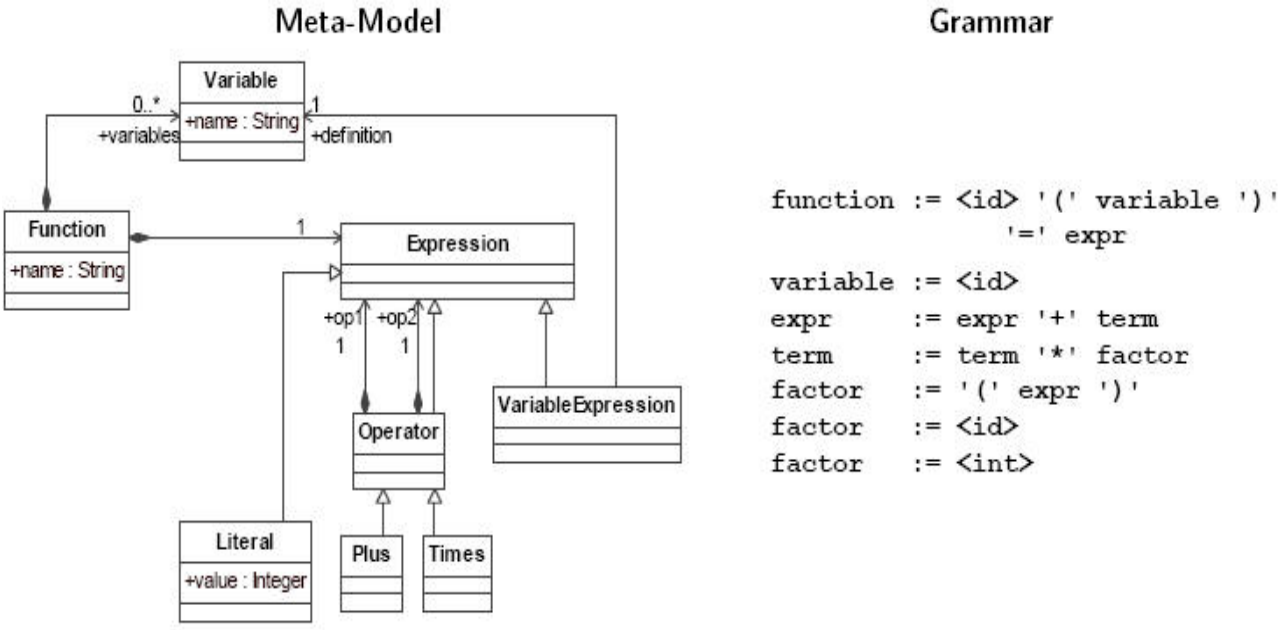
θα πρέπει να έχει ώστε να είναι πλήρως και ορθά ορισμένο. Συνήθη εύρη πληθικότητας είναι 0..1 για μια προεραιτική και 1..1 για μια υποχρεωτική σύνδεση, 0..* για οποιοδήποτε αριθμό συνδέσεων, 1..* για τουλάχιστον μια σύνδεση, όλα πολύ όμοια με τα EBNF εύρη επανάληψης. Μια ιδιότητα έχει επίσης κάποια άλλα χαρακτηριστικά: τα στοιχεία της μπορεί να είναι διατεταγμένα (ordered), μπορεί να επαναλαμβάνονται πολλαπλές φορές (bag) ή μπορεί να δηλώνουν ιδιοκτησία (composition). Κάθε απαρίθμηση περιλαμβάνει μια λίστα από τιμές που περιγράφονται από συμβολοσειρές (strings). Στιγμιότυπα από μια απαρίθμηση περιλαμβάνουν ακριβώς μια τιμή. Μια απαρίθμηση δεν περιέχει ιδιότητες και δεν έχει υπο ή υπέρκλάσεις.

Ακολουθώντας την παραπάνω προσέγγιση, η χρήση του μεταμοντέλου δίνει την δυνατότητα ανάπτυξης περισσότερων τους ενός διακριτών αναπαραστάσεων. Οι γλώσσες μοντελοποίησης μπορούν επίσης να βασίζονται σε άλλες αναπαραστάσεις, όπως πίνακες, φόρμες είναι να αναπαρίστανται υπό μορφή κειμένου. Παρόλα αυτά ένας από τους πιο συχνούς τομείς αποτυχίας είναι ο συμβολισμός της γλώσσας – τα σύμβολα και τα εικονίδια. Για το σκοπό αυτό ο σχεδιαστής της γλώσσας θα πρέπει να ακολουθεί καθιερωμένες συμβάσεις ονοματολογίας και να δίνει ένα ακριβό ορισμό για κάθε έννοια μοντελοποίησης (modeling concept). Η γλώσσα θα πρέπει να είναι όσο το δυνατόν απλούστερη χωρίς περιττές έννοιες και να υπάρχει η δυνατότητα για πιθανές επεκτάσεις αυτής με άμεσο τρόπο. Ας σημειωθεί ότι μοντέλα και μετάμοντέλα μπορούν να αποθηκευτούν σε αποθετήρια (repositories), χρησιμοποιώντας την XMI (XML Metadata Interchange) μορφή [46] και να χρησιμοποιηθούν μελλοντικά σε διάφορες επεκτάσεις και συνδυασμούς.

3.1.1.1 Διαφορές μεταξύ ενός μεταμοντέλου και μιας γραμματικής

Όπως είδαμε και στο κεφάλαιο 2, η πιο συχνά χρησιμοποιούμενη μεταγλώσσα για τον ορισμό γλωσσών προγραμματισμού είναι η EBNF ενώ το κύριο πρότυπο για τον ορισμό μεταμοντέλων είναι το MOF. Τα χαρακτηριστικά των MOF-like πλατφόρμων είναι μια γλώσσα ορισμού με αντικειμενοστραφή σύνταξη και ένα σημασιολογικό πεδίο το οποίο είναι συγκρίσιμο με κατευθυνόμενους γράφους με ετικέτες (directed labeled graphs). Η αντικειμενοστραφής φύση επιτρέπει υψηλή άρθρωση και επαναχρησιμοποίηση μετα-μοντέλων ενώ το σημασιολογικό αυτό πεδίο επιτρέπει την κάλυψη γραφικών γλωσσών (γράφοι) καθώς επίσης και γλωσσών υπό μορφή κειμένου (δέντρα). Η γενικότητα των γράφων σα δομές δεδομένων επίσης προτείνει ότι το MOF είναι εφαρμόσιμο σε άλλες πιθανόν ανεξερεύνητες μορφές γλωσσών. Η κύρια λοιπόν διαφορά μεταξύ

μεταμοντέλων και γραμματικών είναι ότι τα μεταμοντέλα περιγράφουν γράφους, ενώ οι γραμματικές περιγράφουν δέντρα. Για παράδειγμα υπάρχουν ορισμένες συνδέσεις στο μοντέλο που δεν αναπαριστούνται απευθείας στο δέντρο συντακτικής ανάλυσης. Η εικόνα 2 αναπαριστά την αφηρημένη σύνταξη μιας πολύ απλής γλώσσας περιγραφής μαθηματικών εκφράσεων τόσο υπό μορφή μεταμοντέλου όσο και υπό μορφή γραμματικής. Στο παράδειγμα αυτό, υπάρχουν συνδέσεις μεταξύ των στιγμιότυπων «VariableExpression» και «Variable» οι οποίες δημιουργούν έναν κύκλο στο γράφο του μεταμοντέλου.



Εικόνα 2: Παράδειγμα περιγραφής αφηρημένης σύνταξης μιας γλώσσας υπό μορφή μεταμοντέλου και γραμματικής.

Σύμφωνα με το [25], άλλες σημαντικές διαφορές είναι η αδυναμία ταξινόμησης (ordering) των χαρακτηριστικών μιας μετακλάσης, έναντι της ταξινόμησης των στοιχείων στο δεξιό μέρος του EBNF κανόνα για ένα μη τερματικό σύμβολο. Επίσης οι περισσότερες κανόνες γραμματικής περιλαμβάνουν μία ή περισσότερες λέξεις κλειδιά, ενώ το μεταμοντέλο όχι. Αυτές οι λέξεις κλειδιά είναι σχετικές με τον parser επειδή βοηθούν τον parser να διαφοροποιήσει μεταξύ των στοιχείων της γλώσσας (language elements).

3.1.1.2 Αντιστοιχία ενός μεταμοντέλου σε μια γραμματική

Στο [29], μελετάται η σχέση μεταξύ γραμματικών χωρίς συμφραζόμενα (BNF φορμαλισμός) και μεταμοντέλων σε μορφή MOF και αναγνωρίζονται το πότε και το πώς

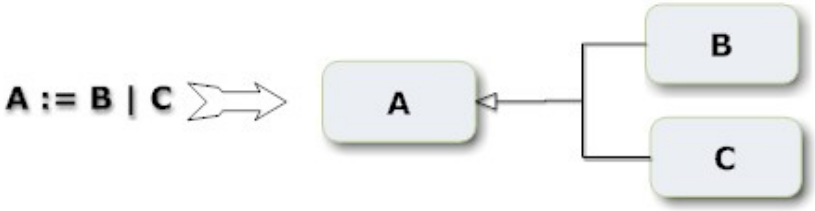
μπορούμε να μετατρέψουμε μια γραμματική σε ένα μεταμοντέλο και το αντίστροφο. Εξάλλου η ιδέα της παραγωγής μεταμοντέλων σε άλλες υπάρχοντες περιγραφές δεν είναι μια καινούρια ιδέα. Σύμφωνα με την βιβλιογραφία [9] [12], για την αντιστοιχία μεταξύ EBNF εννοιών και MOF εννοιών μπορούν να εφαρμοστούν ορισμένοι βασικοί κανόνες όπως:

- αναπαράσταση του αριστερού μέρους ενός κανόνα παραγωγής ως κλάση. Τα στοιχεία του δεξιού μέρους του κανόνα συνδέονται με τη κλάση του αριστερού μέρους με μια συσχέτιση σύνθεσης.

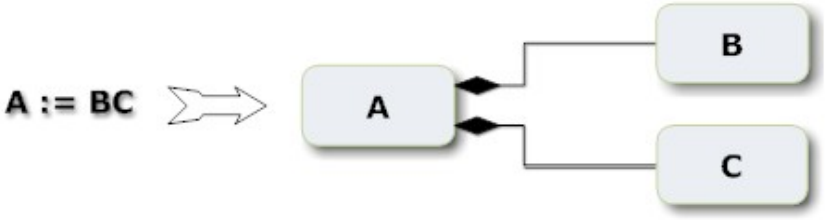
- αναπαράσταση κάθε επανάληψης από μία «μηδέν ή περισσότερα» συσχέτιση.

- αναπαράσταση κάθε προαιρετικού στοιχείου από μία προαιρετική συσχέτιση.

- αναπαράσταση κάθε επιλογής ως υποκλάση της κλάσης του αριστερού μέρους.



- αναπαράσταση μια ακολουθίας με ξεχωριστές συσχετίσεις σύνθεσης.



Ωστόσο, η περιγραφή μιας αντιστοιχίας από μεταμοντέλα σε EBNF γραμματικές είναι σε πολλές περιπτώσεις πιο απαιτητική από ότι το ανάποδο. Ο λόγος είναι ότι τα μεταμοντέλα έμφυτα περιλαμβάνουν περισσότερη πληροφορία από ότι οι EBNF γραμματικές. Εξάλλου μια EBNF γραμματική σχηματίζει ένα δέντρο ενώ τα μεταμοντέλα

σχηματίζουν γράφους με ειδικές ακμές οι οποίες μπορούν να ερμηνευτούν με διάφορος τρόπους.

Σύμφωνα με το [7], ο πίνακας 2 συνοψίζει τους κυριότερους φορμαλισμούς ορισμού γλωσσών και περιγράφει την αντιστοιχία των διαφόρων εννοιών και του σημασιολογικού τους πεδίου. Οι γραμματικές χρησιμοποιούνται κυρίως για γλώσσες υπό μορφή κειμένου, οι MOF-like πλατφόρμες πηγάζουν από την ανάπτυξη λογισμικού βασισμένη σε μοντέλα ενώ η XML χρησιμοποιείται για ανταλλαγή πληροφορίας και διαλειτουργικότητά.

Πίνακας 2: Αντιστοιχίες μεταξύ των κυριοτέρων φορμαλισμών ορισμού γλωσσών.

BNF γλώσσα	γλώσσα για περιγραφή XML σχήματος	μοντέλο MOF
BNF γραμματική	XML σχήμα	μεταμοντέλο
προτάσεις αλφαβήτου	XML κείμενα	μοντέλο
συντακτικό δέντρο, παραγωγές	Document Object Model (DOM)	κατευθυνόμενος γράφος με ετικέτες

3.1.2 Ο γεννήτορας

Σε μια ολοκληρωμένη λύση βασισμένη σε μοντέλα ο γεννήτορας αποτελεί τον ακρογωνιαίο λίθο καθώς λαμβάνει υπόψη του τη γλώσσα μοντελοποίησης, το πλαίσιο εκτέλεσης (domain framework) και το αποτέλεσμα που πρέπει να παραχθεί. Σε αρκετές περιπτώσεις, η κατασκευή του γεννήτορα είναι ένα από τα πιο απαιτητικά κομμάτια μια DSM λύσης καθώς απαιτεί πολύ καλή κατανόηση όλων των παραπάνω. Οι γεννήτορες παράγουν μια μεγάλη ποικιλία από διαφορετικά είδη εξόδου: απλό κείμενο, αρχεία τεκμηρίωσης, script αρχεία, αρχεία ρυθμίσεων (configuration files), XML και φυσικά διάφορα είδη πηγαίου κώδικα. Για το λόγο αυτό, πριν κατασκευάσουμε ένα γεννήτορα, είναι ανάγκη να ξέρουμε τι θα παράξουμε. Ο καλύτερος τρόπος για να πετύχουμε αυτό είναι να έχουμε ένα παραδειγματικό μοντέλο και τη σωστή έξοδο που πρέπει να παραχθεί από αυτό το μοντέλο. Για την αντιμετώπιση αυτών των πιθανών δυσκολιών, δύο καλές πρακτικές σχεδίασης γεννητόρων πρέπει να ακολουθούνται στη πράξη:

- μη προσπαθήσεις να κατασκευάσεις τον γεννήτορα πολύ νωρίς χωρίς να έχει ολοκληρωθεί η γλώσσα μοντελοποίησης και να υπάρχει ένα παραδειγματικό μοντέλο με τη σωστή έξοδο, και
- κράτα τον γεννήτορα όσο πιο απλό γίνεται ωθώντας την πολυπλοκότητα κάτω στο πλαίσιο εκτέλεσης.

Μετά την παραγωγή, ο κώδικας δεν απαιτεί χειροκίνητες τροποποιήσεις ή προσθέσεις. Η επεξεργασία παραγόμενου κώδικα είναι (ή θα έπρεπε να είναι) ανάλογη με τη χειροκίνητη τροποποίηση κώδικα μηχανής μετά π.χ. τη μεταγλώττιση κώδικα γραμμένου σε μια γλώσσα υψηλού επιπέδου όπως η C. Η ιδέα της παραγωγής κώδικα σε καμία περίπτωση δεν αποτελεί καινούρια ιδέα. Στη βιβλιογραφία δύο είναι οι κύριες προσεγγίσεις που απαντώνται [1].

Η απλούστερη προσέγγιση είναι βασισμένη σε πρότυπα επισκέπτη (visitor-based) και η οποία κάνει μια αντιστοιχία από δομές της γλώσσας μοντελοποίησης σε δομές της γλώσσας εξόδου. Ο γεννήτορας επισκέπτεται κάθε στοιχείο του μοντέλου, καλώντας τον γεννήτορα γι' αυτό τον τύπο στοιχείου μέσω ενός προτύπου επισκέπτη (visitor pattern). Εναλλακτικά, υπάρχει και η προσέγγιση βασισμένη σε πρότυπα (template-based). Ένα πρότυπο αποτελείται από στατικά κομμάτια κώδικα τα οποία θέλουμε ως έξοδο καθώς και από κομμάτια τα οποία ποικίλουν ανάλογα με τις τιμές των στοιχείων του μοντέλου. Αυτή η προσέγγιση ακολουθείται και στην κατασκευή δυναμικών ιστοσελίδων στην πλευρά του πελάτη (client) με την HTML να περιλαμβάνει JavaScript εντολές, ή στην πλευρά του εξυπηρέτη (server) με χρήση των γλωσσών PHP ή ASP. Οι γεννήτορες αυτής της μορφής ενδεχομένως είναι οι πιο κοινοί και από τους παλιότερα χρησιμοποιούμενους. Παραδείγματα τεχνολογιών βασισμένα σε πρότυπα περιλαμβάνουν τα JET (Java Emitter Templates), τη Microsoft DSL Tools T4 engine και την CodeWorker scripting γλώσσα.

Ανεξάρτητα από το πια προσέγγιση υιοθετείται κάθε φορά, για να είναι εφικτή η αποτελεσματική παραγωγή αποτελέσματος, η έξοδος που απαιτείται από έναν δοθέντα γεννήτορα πάνω από διαφορετικά μοντέλα πρέπει να παρουσιάζει καλά ορισμένα πρότυπα. Τις περισσότερες φορές, ο γεννήτορας δεν είναι ορατός στους χρήστες και μπορεί να θεωρηθεί παρόμοιος με έναν μεταγλωττιστή. Σε κάθε περίπτωση πρέπει να υποστηρίζεται πλήρης παραγωγή κώδικα όπου τόσο στατικές (static) όσο και (behavioral) δομές παράγονται. Επιπλέον, έχοντας ως πηγή το μοντέλο που αναπτύσσεται ο γεννήτρας μπορεί να παράξει πολλαπλά διαφορετικά αποτελέσματα.

3.1.3 Το πλαίσιο εκτέλεσης

Το πλαίσιο εκτέλεσης (domain framework) παρέχει την διεπαφή μεταξύ του παραγόμενου κώδικα και του υποκείμενου περιβάλλοντος στόχου (target environment) ή πλατφόρμας στόχου (target platform). Είναι ένα επίπεδο κώδικα που η κύρια λειτουργία του είναι η αποφυγή της επανάληψης και της πολυπλοκότητας στον παραγόμενο κώδικα και επίσης στον γεννήτορα. Μετακινώντας την ποικιλότητα από το γεννήτορα στο πλαίσιο εκτέλεσης απλοποιείται η διαδικασία πιθανής προσθήκης υποστήριξης για νέες πλατφόρμες. Παρόλα αυτά σε λύση βασισμένη σε μοντέλα η ύπαρξη ενός πλαισίου εκτέλεσης δεν είναι πάντα αναγκαία. Σε πολλές περιπτώσεις η ίδια η πλατφόρμα στόχος μπορεί να παρέχει την κατάλληλη διεπαφή η οποία μπορεί να χρησιμοποιηθεί κατευθείαν από τον γεννήτορα. Ωστόσο πολύ συχνά χρησιμοποιείται κάποιο βοηθητικό πλαίσιο εκτέλεσης για να κάνει τον παραγόμενο κώδικα απλούστερο.

Η ανάπτυξη του κατάλληλου πλαισίου εκτέλεσης είναι μια χειροκίνητη διαδικασία η οποία απαιτεί πολύ καλή γνώση της υποκείμενης πλατφόρμας και του επιθυμητού αποτελέσματος και είναι ευθύνη του σχεδιαστή της DSM λύσης. Το πλαίσιο αυτό χρησιμοποιείται μόνο από το γεννήτορα και συνεπώς λεπτομέρειες χρήσης του δεν είναι ανάγκη να γίνουν γνωστές στους τελικούς χρήστες. Μόνο ένας έμπειρος προγραμματιστής πρέπει να μάθει ή να αναπτύξει το πλαίσιο εκτέλεσης, να το εφαρμόσει στο γεννήτορα και έτσι όλοι οι άλλοι επωφελούνται χρησιμοποιώντας απλά την DSM λύση. Ας σημειωθεί, ότι πολύ συχνά ο προγραμματιστής που αναπτύσσει το πλαίσιο εκτέλεσης είναι και αυτός ο οποίος κατασκευάζει και τον γεννήτορα και επομένως το κόστος ανάπτυξης και συντήρησης γίνεται ακόμα μικρότερο.

3.2 Παράγοντες υιοθέτησης μιας γλώσσας μοντελοποίησης ειδικού σκοπού

Ορισμένες τυπικές περιπτώσεις για υιοθέτηση μιας ολοκληρωμένης λύσης βασισμένης σε μοντέλα είναι εταιρείες ή οργανισμοί που διαθέτουν μια γραμμή παραγωγής, κατασκευάζουν παρόμοια είδη προϊόντων ή εφαρμογές πάνω από μια κοινή βιβλιοθήκη ή πλατφόρμα. Ωστόσο, τα οφέλη από μια τέτοια λύση δεν είναι χωρίς κόστος. Κάποιος πρέπει να δημιουργήσει τη DSM λύση και αυτό απαιτεί την αρχική επένδυση. Το αρχικό αυτό κόστος περιλαμβάνει σχεδόν αποκλειστικά τον ανθρώπινο παράγοντα καθώς απαιτούνται ειδικοί του πεδίου και έμπειροι προγραμματιστές. Παλαιότερα το κόστος ανάπτυξης των εργαλείων ήταν επίσης σχετικά υψηλό καθώς οι εταιρείες απαιτούνταν να κατασκευάσουν τα εργαλεία από την αρχή ή χρησιμοποιώντας υποστηρικτικές βιβλιοθήκες και πλαίσια. Επιπλέον υπάρχει το κόστος συντήρησης και αναβάθμισης.

Αν το πεδίο είναι στατικό η εταιρεία συνεχίζει να χρησιμοποιεί την αρχική DSM λύση χωρίς τροποποιήσεις - κάτι πολύ σπάνιο - και με ελάχιστο κόστος συντήρησης. Είναι όμως πολύ πιθανό ότι το πεδίο εξελίσσεται ή η εταιρεία βρίσκει καλύτερους τρόπους να αυτοματοποιεί την ανάπτυξη εφαρμογών χρησιμοποιώντας τη λύση αυτή.

Για το λόγο αυτό, αρκετά πράγματα πρέπει να ληφθούν υπόψη για την ανάπτυξη μιας ολοκληρωμένης λύσης βασισμένης σε μοντέλα και κατά συνέπεια και μια γλώσσας μοντελοποίησης ειδικού σκοπού. Αρχικά, πρέπει να εξεταστεί αν υπάρχει ήδη ένα μοντέλο το οποίο είναι αρκετά κοντά στο πεδίο του προβλήματος. Αν υπάρχει, τότε εξετάζεται αν το υπάρχον μοντέλο μπορεί να επεκταθεί ή να βελτιωθεί. Επίσης, πρέπει να ληφθεί σοβαρά υπόψη αν είναι ανάγκη το μοντέλο να βασίζεται πάνω σε κάποιο standard και να φτιαχτεί πάνω από τη γλώσσα μοντελοποίησης μία ή περισσότερες γραμμές παραγωγής λογισμικού (software product line). Έπειτα, αν υπάρχει ανάγκη για διαφορετικές αναπαραστάσεις (γραφική αναπαράσταση, υπό μορφή κειμένου κ.α) και επεξεργασία της γλώσσας μοντελοποίησης. Αντίστοιχα πρέπει να εξεταστεί και η απαίτηση για την ανάπτυξη των κατάλληλων υποστηρικτικών εργαλείων ή πιθανή επέκταση κάποιων προϋπαρχόντων. Τέλος, αν απαιτείται μοντελοποίηση δυναμικής συμπεριφοράς (dynamic behavior), πρέπει να μελετηθούν οι τρόποι με τους οποίους μπορεί να επιτευχθεί αυτό αποτελεσματικά.

Για τους σκοπούς λοιπόν αυτούς κάποιες καλές πρακτικές που απαντώνται στη βιβλιογραφία προτείνουν την εκμετάλλευση υπαρχόντων μοντέλων όπου αυτό είναι δυνατόν. Ορισμοί XML σχημάτων (XML Schema Definition - XSD) και ορισμοί EMF μοντέλων είναι δυο πολύ διαδεδομένες τεχνολογίες που μπορούν να χρησιμοποιηθούν για την αναζήτηση υπαρχόντων μοντέλων που εφαρμόζονται σε συγκεκριμένα πεδία και πιθανόν μπορούν να φανούν πολύτιμες πριν την ξαναανακάλυψη του τροχού. Ας σημειωθεί ότι σχεδόν οποιοδήποτε XSD μπορεί να μετατραπεί στον αντίστοιχο EMF ορισμό μοντέλου διευκολύνοντας έτσι ακόμα περισσότερο την επαναχρησιμοποίηση υπαρχόντων μοντέλων.

Επίσης σύμφωνα με τους συγγραφείς του [20] υπάρχουν κάποιες συχνές απαιτήσεις για την ανάπτυξη γλωσσών μοντελοποίησης ειδικού σκοπού. Ένα μοντέλο πρέπει πάντα να βρίσκεται σε ένα υψηλότερο επίπεδο αφαίρεσης από ότι ο παραγόμενος κώδικας. Στόχος δεν είναι ο προγραμματισμός με εικόνες. Αυτό υπονοεί ότι σε μια DSL χρησιμοποιούνται έννοιες (concepts) μόνο όταν η έννοια είναι ευκολότερο να μοντελοποιηθεί από ότι να γραφεί προγραμματιστικά. Πράγματα τα οποία απαιτούν τόση προσπάθεια να μοντελοποιηθούν όσο και γραφούν προγραμματιστικά δεν πρέπει

να ενσωματώνονται στη DSL. Μια άμεση συνέπεια του παραπάνω είναι ότι συχνά δεν μοντελοποιείται 100% μια εφαρμογή. Ένα μέρος του συστήματος μοντελοποιείται και ένα μέρος παράγεται προγραμματιστικά. Εξάλλου, πολλές φορές ο γεννήτορας σχεδιάζεται με τέτοιο τρόπο ώστε ο κώδικας που γράφεται με το χέρι να προστίθεται ξεχωριστά από τον αυτόματα παραγόμενο κώδικα. Επίσης, τα μοντέλα πρέπει να καθιστούν τους λιγότερο έμπειρους προγραμματιστές με περιορισμένη αρχιτεκτονική γνώση πιο παραγωγικούς. Αυτό επιτυγχάνεται κρύβοντας τις τεχνικές λεπτομέρειες πίσω από τα μοντέλα με άμεση συνέπεια την όσο το δυνατόν μεγαλύτερη απλότητα του μοντέλου.

Ο παραγόμενος κώδικας πρέπει να είναι ευανάγνωστος και συντηρήσιμος. Η σχέση μεταξύ του μοντέλων και παραγόμενου κώδικα πρέπει να είναι σαφής, και ειδικότερα για τους πιο έμπειρους προγραμματιστές οι οποίοι έχουν την ευθύνη της προσθήκης χειρόγραφου κώδικα στον παραγόμενο κώδικα. Κάθε στοιχείο της γλώσσας έχει μια ρητή σύνδεση με ένα ή περισσότερα δομικά στοιχεία/κομμάτια και παράγει κώδικα για ακριβώς αυτά τα κομμάτια. Αυτή η αυτόματη παραγωγή κώδικα συνεπάγεται αύξηση της παραγωγικότητας και της ποιότητας του τελικού αποτελέσματος.

3.3 Παραδείγματα υιοθέτησης DSM λύσεων

Στα [1] και [59] παρουσιάζονται με αρκετές λεπτομέρειες κάποια παραδείγματα υιοθέτησης ολοκληρωμένων λύσεων βασισμένων σε μοντέλα. Κάθε πεδίο είναι διαφορετικό, και άρα κάθε παράδειγμα είναι διαφορετικό. Τα πεδία προβλήματος ποικίλλουν καθώς και τα περιβάλλοντα στόχοι. Τα τελευταία χρόνια, μεγάλες εταιρείες και οργανισμοί καταφεύγουν σε τέτοιου είδους λύσεις προσπαθώντας να δημιουργήσουν κάποια γραμμή παραγωγής για τα προϊόντα που αναπτύσσουν. Απώτερος στόχος είναι η αύξηση της παραγωγικότητας και ποιότητας του τελικού προϊόντος σε συνδυασμό με το όσο το δυνατό μικρότερο χρόνο διάθεσης του προϊόντος στην αγορά. Ειδικότερα στα πεδία του κινητού υπολογισμού και των ενσωματωμένων συστημάτων όπου τα τελευταία χρόνια οι εξελίξεις είναι ραγδαίες, η υιοθέτηση τέτοιων λύσεων είναι επιτακτικές. Ορισμένα ενδεικτικά παραδείγματα τέτοιων ολοκληρωμένων λύσεων βασισμένων σε μοντέλα βρίσκουν ήδη εφαρμογή σε πεδία όπως

- IP τηλεφωνία και επεξεργασία κλήσεων (πρωτόκολλα SIP και H.232)
- ανάπτυξη επιχειρηματικών εφαρμογών για κινητά τηλέφωνα χρησιμοποιώντας ένα Python πλαίσιο (Nokia)

- ανάπτυξη διεπαφών για όργανα μέτρησης για αθλητές (π.χ. παρακολούθηση καρδιακών παλμών) (Polar Electro)
- ανάπτυξη ιατρικών εφαρμογών (Philips)
- οικιακός αυτοματισμός και ενσωματωμένα συστήματα

3.4 Περιβάλλοντα ανάπτυξης εργαλείων μοντελοποίησης

Ένα πολύ σημαντικό κομμάτι για την επιτυχία της υιοθέτησης μιας νέας DSM λύσης από τους τελικούς χρήστες είναι και η καλή υποστήριξη εργαλείων για την επεξεργασία και τον έλεγχο των εφαρμογών που πρόκειται να αναπτυχθούν. Πιθανή είναι η δημιουργία ενός εργαλείου μοντελοποίησης από την αρχή αλλά ακόμη και για κάποιες από τις μεγαλύτερες ομάδες προγραμματιστών μια τέτοια προσέγγιση θα είναι απαγορευτικά ακριβή και χρονοβόρα [16]. Εύκολα λοιπόν γίνεται κατανοητό ότι υπάρχει η ανάγκη για αποδοτικούς τρόπους παραγωγής τέτοιων εργαλείων είτε με αυτόματο ή ημιαυτόματο τρόπο. Τα λεγόμενα Language Workbences [34] στοχεύουν προς αυτή την κατεύθυνση στηρίζοντας τον σχεδιαστή της γλωσσάς σε όλα τα στάδια ανάπτυξης μιας ολοκληρωμένης DSM λύσης, από τον ορισμός της γλώσσας μοντελοποίησης έως και την παραγωγή των κατάλληλων υποστηρικτικών εργαλείων.

Σύμφωνα και με το DSM forum [59], προς αυτή την κατεύθυνση ένα μεγάλο πλήθος από τέτοια ολοκληρωμένα περιβάλλοντα ανάπτυξης έχουν αναπτυχθεί και εξελίσσονται από διάφορους οργανισμούς. Στην αγορά, υπάρχουν ήδη διαθέσιμα εμπορικά εργαλεία από μεγάλες εταιρείες καθώς και εργαλεία και πρωτότυπα που αναπτύσσονται από ερευνητικούς φορείς και κοινότητες ανοικτού λογισμικού. Μια ολοκληρωμένη ανασκόπηση και σύγκριση αυτών των εργαλείων ανάπτυξης είναι εκτός του σκοπού αυτής της διπλωματικής εργασίας αλλά ωστόσο θα παρουσιάσουμε συνοπτικά τα πιο αντιπροσωπευτικά από κάθε κατηγορία.

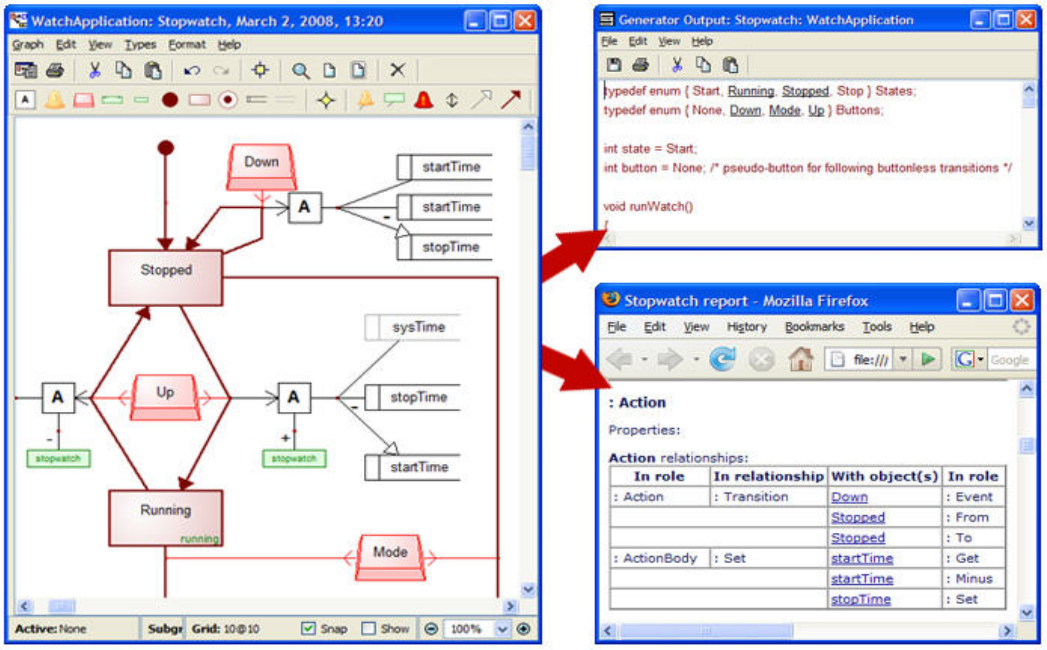
3.4.1 Το περιβάλλον MetaEdit+

Το MetaEdit+ [60] είναι ένα εμπορικό metaCASE εργαλείο που αναπτύσσεται από τη Φιλανδική εταιρεία MetaCase Consulting. Αποτελεί τον απόγονο του προγενέστερου εργαλείου MetaEdit ως προς τις ιδέες και την ομάδα ανάπτυξης του, αλλά στην πράξη αποτελεί ένα φρέσκο ξεκίνημα ως προς την υλοποίηση του. Σύμφωνα με τις γνώμες πολλών ειδικών της βιομηχανίας όπως ο Scott Ambler και ο Markus Volter, το MetaEdit+ είναι το πιο ώριμο και προηγμένο DSM περιβάλλον. Στα πλαίσια ανάπτυξης

του έχουν εκπονηθεί περίπου 30 ερευνητικές εργασίες και περισσότερα από 150 άρθρα έχουν δημοσιευθεί.

Στο περιβάλλον του MetaEdit+, δύο είναι τα βασικά συστατικά: το MetaEdit+ Workbench και το CASE εργαλείο. Το MetaEdit+ Workbench χρησιμοποιείται από τον σχεδιαστή της γλώσσας και του παρέχει όλα τα απαραίτητα μέσα για τον ορισμό της νέας γλώσσας μοντελοποίησης. Για την περιγραφή των εννοιών της γλώσσας χρησιμοποιείται μια απλή αλλά πολύ ισχυρή γλώσσα περιγραφής, η GOPPRR (Graph, Object, Property, Port, Relationship, Role) με την οποία δηλώνονται οι αντίστοιχοι τύποι των εννοιών που ορίζονται στη γλώσσα μοντελοποίησης. Μέσω της κατάλληλης σουίτας εργαλείων, ο σχεδιαστής μπορεί εύκολα να ορίσει τις έννοιες (concepts) της γλώσσας, τις ιδιότητές τους, τα αντίστοιχα σύμβολα και περιορισμούς. Η εκτεταμένη βιβλιοθήκη από επαναχρησιμοποιούμενα συστατικά μοντελοποίησης γλωσσών (modeling language components) σου επιτρέπουν να ξεκινήσεις με έναν πολύ άμεσο τρόπο.

Το εργαλείο CASE χρησιμοποιεί ως είσοδο τον ορισμό της γλώσσας όπως ορίστηκε στο Workbench και παράγει ένα εργαλείο μοντελοποίησης για τον ορισμό αυτό χωρίς να χρειαστεί να γράψεις ούτε μια απλή γραμμή κώδικα. Με τον τρόπο αυτό η διαδικασία ανάπτυξης εργαλείων μοντελοποίησης καθίσταται γρήγορη, διαισθητική και αποδοτική οικονομικά.



Εικόνα 3: Χρήση της γλώσσας μοντελοποίησης, παραγωγή κώδικα και τεκμηρίωσης [60].

3.4.2 Τα Microsoft DSL Tools

Στο Visual Studio 2005 SDK 3.0, η εταιρεία Microsoft κυκλοφόρησε την πρώτη έκδοση των Domain-Specific Language Tools [61]: ένας συνδυασμός από πλαίσια (frameworks), γλώσσες, επεξεργαστές (editors), γεννήτορες (generators) και wizards που επιτρέπουν στους σχεδιαστές γλωσσών να καθορίσουν τις δικές τους γλώσσες μοντελοποίησης και εργαλεία. Αρχικά με το Software Factories project και τώρα με τα Microsoft DSL Tools η Microsoft παρακολουθεί τις εξελίξεις στον τομέα της ανάπτυξης λογισμικού βασισμένου σε μοντέλα, με τον Bill Gates να ισχυρίζεται ότι η προσέγγιση αυτή θα είναι η πιο σημαντική καινοτομία στο λογισμικό για τα επόμενα 10 χρόνια [13].

Χρησιμοποιώντας τα εργαλεία αυτά, ο χρήστης μπορεί να ορίσει με αρκετά διαισθητικό τρόπο τη δική του γραφική γλώσσα μοντελοποίησης ειδικού σκοπού (Visual Domain Specific Language - VDSL) καθώς και ένα βασικό εργαλείο για την επεξεργασία των στιγμιότυπων της γλώσσας αυτής. Επιπρόσθετα, για τη μετατροπή των δομών της γλώσσας μοντελοποίησης σε κάποιου είδους εκτελέσιμη μορφή κώδικα χρησιμοποιείται μια γλώσσα προτύπων (template language), που αποτελείται κυρίως από C# ή Visual Basic δομές. Ένα βασικό μειονέκτημα όλων των παραπάνω εργαλείων μοντελοποίησης είναι ότι δουλεύουν ως μέρος του Visual Studio και άρα είναι διαθέσιμα μόνο για περιβάλλοντα Windows.

3.4.3 Το Generic Modeling Environment

Ένα από τα πιο γνωστά ερευνητικά εργαλεία ανάπτυξης μιας λύσης βασισμένης σε μοντέλα είναι το Generic Modeling Environment (GME) [63]. Το περιβάλλον αυτό έχει σχεδιαστεί και αναπτυχθεί από το πανεπιστήμιο του Vanderbilt στη Γερμανία και όπως και το MetaEdit+ ωρίμασε από ένα ισχυρό υπόβαθρο από προηγούμενη έρευνα και πρακτική εμπειρία στον τομέα των γλωσσών και εργαλείων μοντελοποίησης.

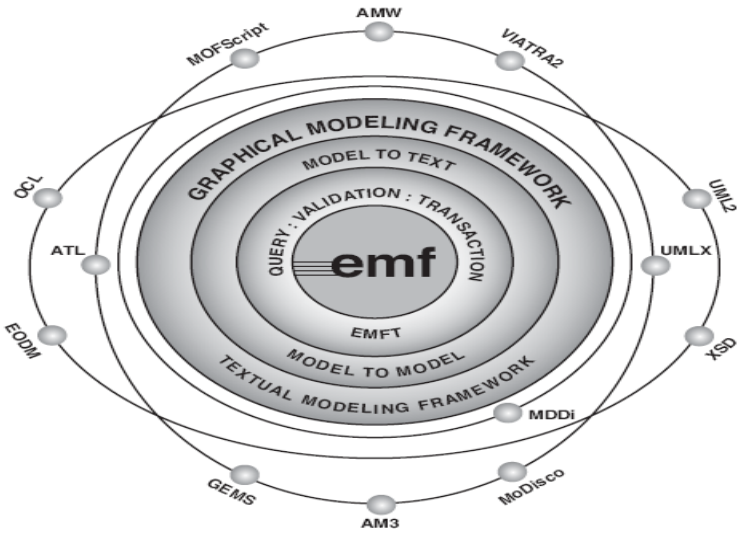
Στο περιβάλλον αυτό, τα μεταμοντέλα ορίζονται σε μια απλή UML-like γλώσσα, η οποία κάνει εξονυχιστική χρήση των stereotypes για να διακρίνει τους διάφορους τύπους των εννοιών της γλώσσας ενώ οι περιορισμοί καθορίζονται σε μια διάλεκτο της OCL (Object Constraint Language) [39] όπου ένας ενσωματωμένος στο περιβάλλον διαχειριστής περιορισμών επιβάλλει όλους τους περιορισμούς του πεδίου κατά την διάρκεια κατασκευής του μοντέλου. Παρέχεται επίσης η δυνατότητα για επαναχρησιμοποίηση και σύνθεση μεταμοντέλων από υπάρχουσες γλώσσες μοντελοποίησης. Στα μειονεκτήματα του συγκαταλέγεται η μη υποστήριξη γραφικού επεξεργαστή για την επεξεργασία των

συμβόλων που αντιστοιχίζονται στις διάφορες έννοιες του πεδίου καθώς και η διαθεσιμότητα του μόνο για περιβάλλοντα Windows.

Παρόλα αυτά το GME διαθέτει μια αρθρωτή, επεκτάσιμη αρχιτεκτονική που χρησιμοποιεί την τεχνολογία MS COM. Κάποιος μπορεί πολύ εύκολα να αναπτύξει περιφερειακά συστατικά και εργαλεία χρησιμοποιώντας οποιαδήποτε γλώσσα προγραμματισμού υποστηρίζει COM (π.χ C++, Visual Basic, C#, Python κ.α).

3.4.4 Το Eclipse Modeling Project

Το Eclipse Modeling Project (EMP) [2, 49] είναι ένα σχετικά νέο top-level project του Eclipse το οποίο αποτελείται από μια συλλογή από projects που σχετίζονται με τεχνολογίες για την ανάπτυξη λογισμικού βασισμένου σε μοντέλα (εικόνα 4). Οργανώνεται λογικά σε projects, υποστηρίζοντας ολόκληρη τη διαδικασία ανάπτυξης μιας ολοκληρωμένης λύσης μοντελοποίησης. Η ανοικτή φύση της πλατφόρμας του Eclipse και οι δυνατότητες για την ανάπτυξη αφηρημένης σύνταξης, διακριτής σύνταξης και μετατροπής μοντέλων που αυτό προσθέτει, το έχουν μετατρέψει στον ελβετικό σουγιά για όσους ασχολούνται με το state-of-the-art στο πεδίο της ανάπτυξης βασισμένης σε μοντέλα.



Εικόνα 4: Το Eclipse Modeling Project [2].

3.4.4.3 Ανάπτυξη αφηρημένης σύνταξης

Ο πυρήνας μιας γλώσσας μοντελοποίησης ειδικού σκοπού είναι η αφηρημένη σύνταξη της και τυπικά το πρώτο στοιχείο της γλώσσας που πρέπει να αναπτυχθεί. Στα πλαίσια

του EMP για το σκοπό αυτό χρησιμοποιείται το Eclipse Modeling Framework (EMF) [3, 50]. Το EMF παρέχει το δικό μεταμοντέλο, που ονομάζεται Ecore και σύμφωνα με το οποίο περιγράφονται οι έννοιες που μοντελοποιεί η γλώσσα. Είναι ένα αρκετά όμοιο μεταμοντέλο με το EMOF πρότυπο (ένα υποσύνολο του MOF 2.0) [45,62] και αρκετά απλό γεγονός που συνεισφέρει και στην ισχύ και δύναμη που αυτό προσφέρει. Ένα μοντέλο ορισμένο σε όρους του Ecore είναι αρκετά δηλωτικό και μπορεί να έχει έναν αριθμό από διαφορετικές διακριτές συντάξεις.

3.4.4.4 Ανάπτυξη διακριτής σύνταξης

Ένα από τα βασικά πλεονεκτήματα του EMP έναντι των υπολοίπων διαθέσιμων περιβαλλόντων ανάπτυξης εργαλείων μοντελοποίησης είναι τα προηγμένα χαρακτηριστικά του για την ανάπτυξη διακριτής σύνταξης (concrete syntax development). Το EMP περικλείει δυο αντίστοιχα πλαίσια (frameworks) τόσο για την ανάπτυξη γραφικών αναπαραστάσεων (graphical representation) των μοντέλων όσο και για την αναπαράσταση υπό μορφή κειμένου (textual representation).

Το Graphical Modeling Framework (GMF) [52] είναι μια πολλά υποσχόμενη open-source τεχνολογία η οποία βασίζεται πάνω στο EMF και το Graphical Editing Framework (GEF) [51]. Βασικός του σκοπός είναι ο ορισμός και η υλοποίηση γλωσσών μοντελοποίησης ειδικού σκοπού. Το GEF είναι ένα πλαίσιο βασισμένο στην MVC (Model View Controller) αρχιτεκτονική και χρησιμοποιείται για την ανάπτυξη γραφικών επεξεργαστών (graphical editors). Τα EMF και GEF δεν είναι ολοκληρωμένα (integrated) και το GMF γεφυρώνει το μεταξύ τους χάσμα παρέχοντας ένα σύνολο από εργαλεία που επιτρέπουν στο σχεδιαστή της γλώσσας να ορίσει και αυτόματα παράξει ένα γραφικό εργαλείο μοντελοποίησης το οποίο ενσωματώνεται στο περιβάλλον του Eclipse. Στην πράξη, το GMF είναι ένα επεκταμένο και βελτιωμένο GEF που έχει αναπτυχθεί από την IBM και χρησιμοποιεί GEF γεννήτορες αναπτυγμένους από την Borland.

Χρησιμοποιώντας το GMF και έχοντας ορίσει την αφηρημένη σύνταξη της γλώσσας ο σχεδιαστής μπορεί εύκολα να αναπτύξει το γραφικό συμβολισμό και να τον αντιστοιχίσει με τις έννοιες και δομές της γλώσσας. Η δημιουργία ενός απλού γραφικού επεξεργαστή για την επεξεργασία στιγμιότυπων της γλώσσας ακολουθεί μια καλά προκαθορισμένη διαδικασία η οποία συνοψίζεται στην δημιουργία πέντε διαφορετικών XML αρχείων, τα περισσότερα από τα οποία μπορούν να δημιουργηθούν βήμα προς βήμα από γραφικά εργαλεία και διαλόγους (dialogs). Αρχικά ορίζονται όλες οι έννοιες της γλώσσες που θα

αναπαρασταθούν γραφικά καθώς και ο γραφικός συμβολισμός τους. Ακολουθεί ο ορισμός του τρόπου με τον οποίο θα αναπαρίστανται οι διάφορες έννοιες στην παλέτα εργαλείων του γραφικού επεξεργαστή και έπειτα συνδυάζονται με τις έννοιες της αφηρημένης σύνταξης της γλώσσας. Τέλος, ακολουθεί η αυτόματη παραγωγή του κώδικα που υλοποιεί τον γραφικό επεξεργαστή και τον ενσωματώνει κατάλληλα στο περιβάλλον του Eclipse.

Αντίστοιχα, για την ανάπτυξη διακριτής σύνταξης υπό μορφή κειμένου και για την παραγωγή ενός εργαλείου επεξεργασίας των στιγμιότυπων της γλώσσας υπό αυτή την μορφή μπορεί να χρησιμοποιηθεί το Textual Modeling Framework (TMF) [55]. Το πλαίσιο αυτό είναι ένα σχετικά νέο πολλά υποσχόμενο project του Eclipse το οποίο χρησιμοποιώντας τις εμπειρίες και τη γνώση που έχει αποκτηθεί από άλλα διάσημα πλαίσια όπως το openArchitectureWare [57] επιχειρεί να γεφυρώσει το χάσμα μεταξύ των παραδοσιακών τεχνικών ανάπτυξης γλωσσών υπό μορφή κειμένου και της ανάπτυξης βασισμένης σε μοντέλα. Χρησιμοποιείται κυρίως για την παραγωγή εξωτερικών γλωσσών ειδικού σκοπού και στην παρούσα μορφή του παρέχει τη δυνατότητα της παραγωγής υψηλής ποιότητας επεξεργαστών κειμένου με προηγμένες δυνατότητες, οι οποίοι ενσωματώνονται αυτόματα στο περιβάλλον του Eclipse.

3.4.4.5 Μετασχηματισμοί μοντέλων

Η ανάπτυξη μιας γλώσσας ειδικού σκοπού πολύ συχνά αποσκοπεί στην αύξηση του επιπέδου αφαίρεσης στο οποίο επιθυμεί κάποιος να δουλεύει. Εκτός όμως από τον ορισμό μιας γλώσσας ειδικού σκοπού και την παραγωγή των κατάλληλων εργαλείων για την επεξεργασία των στιγμιότυπων της, τυπικά κάποια έξοδος πρέπει να μπορεί να παραχθεί από τα στιγμιότυπα αυτά, διαφορετικά η όλη προσπάθεια δε θα είχε και τόσο μεγάλη πρακτική σημασία. Για το σκοπό αυτό, το EMP παρέχει εργαλεία τόσο για μετασχηματισμό μοντέλο-σε-κείμενο (model-to-text) όσο και μοντέλο-σε-μοντέλο (model-to-model).

Στην κατηγορία εργαλείων μετασχηματισμών μοντέλο-σε-κείμενο, το πιο γνωστό ίσως συστατικό είναι το Java Emitter Templates (JET) [53] το οποίο χρησιμοποιείται και από το ίδιο το EMF για την αυτόματη παραγωγή Java κώδικα. Εναλλακτικά, ένα ολοένα και πιο δημοφιλή εργαλείο είναι το Xpand [53], μια μηχανή προτύπων (template engine), που χρησιμοποιείται σε πάρα πολύ μεγάλο βαθμό από το GMF. Αντίστοιχα εργαλεία για το μετασχηματισμό ενός μοντέλου σε ένα άλλο είναι τα QVT και ATL.

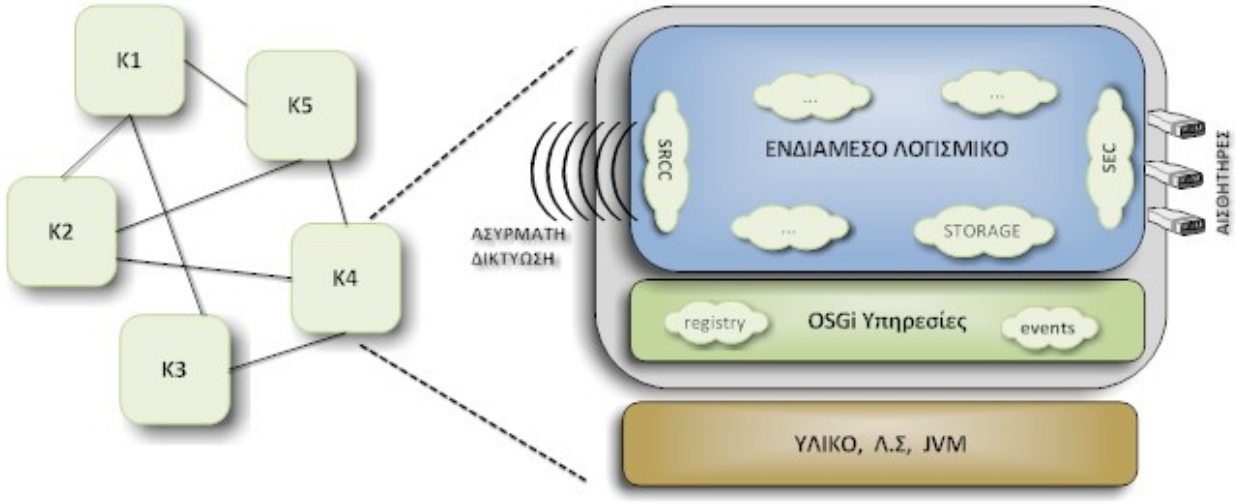
Τέλος, αξίζει να σημειωθεί ότι το EMP περιλαμβάνει και μια πληθώρα άλλων βοηθητικών εργαλείων τα οποία ενισχύουν την ανάπτυξη βασισμένη σε μοντέλα στο περιβάλλον του Eclipse. Για παράδειγμα το Model Development Tools (MDT) Project παρέχει μια μεγάλη γκάμα από εργαλεία που παρέχουν υποστήριξη για βιομηχανικά πρότυπα μοντέλων (industry-standard models) και δυνατότητες όπως εισαγωγή XML schemas και αυτόματη μετατροπή σε EMF μοντέλα, υποστήριξη για το πρότυπο UML2 και της γλώσσας OCL για ορισμό περιορισμών σε μοντέλα κ.α.

ΜΕΡΟΣ Β: ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ

4. ΕΝΑ ΠΕΡΙΒΑΛΛΟΝ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ ΓΙΑ ΤΟ ΔΙΑΧΥΤΟ ΥΠΟΛΟΓΙΣΜΟ

4.1 Διάχυτος υπολογισμός

Ο διάχυτος υπολογισμός ως ιδέα έγινε κυρίως γνωστός από τον Mark Weiser το 1993 [31]. Δεν είναι τίποτε άλλο παρά το όραμα ενός κόσμου γεμάτου από έξυπνες και επικοινωνούσες μεταξύ τους συσκευές, οι οποίες όμως είναι με φυσικό τρόπο ενταγμένες στον περιβάλλοντα χώρο και στα χρηστικά αντικείμενα ενώ εξυπηρετούν τον άνθρωπο με φυσικό τρόπο και δηλώνουν την παρουσία τους όσο το δυνατόν λιγότερο. Σε ένα τυπικό σενάριο ενός συστήματος διάχυτου υπολογισμού πολλές τέτοιες συσκευές αυτό-οργανώνονται σε δίκτυο και επικοινωνούν μεταξύ τους για την επίτευξη κάποιου κοινού στόχου. Στην εικόνα 5 παρουσιάζεται με μεγαλύτερη λεπτομέρεια η τυπική αρχιτεκτονική ενός τέτοιου κόμβου του δικτύου.



Εικόνα 5: Τυπική αρχιτεκτονική ενός κόμβου σε περιβάλλον διάχυτου υπολογισμού.

Στα κατώτερα στρώματα της στρωματοποιημένης αρχιτεκτονικής βρίσκεται το υλικό (hardware) του κόμβου μαζί με την κατάλληλη υποστήριξη από το λειτουργικό σύστημα και ίσως κάποιο είδος εικονικής μηχανής η οποία απαιτείται για την εκτέλεση των εφαρμογών. Έπειτα ακολουθεί το ενδιάμεσο υλικό που βρίσκεται εγκατεστημένο στον κόμβο και το οποίο παρέχει την κατάλληλη διεπαφή μεταξύ των εφαρμογών και της πλατφόρμας εκτέλεσης. Οι δυνατότητες του κόμβου είναι αρκετά περιορισμένες και πολύ καλά καθορισμένες και γι' αυτό το σκοπό παρέχονται ως υπηρεσίες στο επίπεδο των εφαρμογών. Ένας κόμβος διαθέτει ένα συστατικό ασύρματης επικοινωνίας ενώ

μπορεί να χρησιμοποιεί και ένα πλήθος από διαφορετικού τύπου αισθητήρες. Οι αισθητήρες είναι υπεύθυνοι για την συλλογή μετρήσεων από το εξωτερικό περιβάλλον και ελέγχονται από το συστατικό διαχείρισης αισθητήρων του ενδιάμεσου λογισμικού. Τα δεδομένα αυτά επεξεργάζονται από τον ίδιο τον κόμβο και συχνά προωθούνται και στο υπόλοιπο δίκτυο μέσω του συστατικό διαχείρισης επικοινωνίας του ενδιάμεσου λογισμικού και της διεπαφής ασύρματης επικοινωνίας.

Σε περιβάλλοντα διάχυτου υπολογισμού, όπου υπάρχει μεγάλο πλήθος από τέτοιες έξυπνες συσκευές με προηγμένα χαρακτηριστικά επικοινωνίας και διαχείρισης αισθητήρων μπορεί κανείς να αναπτύξει προηγμένες συνεργατικές εφαρμογές με συναίσθηση του περιβάλλοντος και της τρέχουσας κατάστασης (*context aware*). Ωστόσο, η ανάπτυξη τέτοιων καινοτομικών εφαρμογών απαιτεί πολύ εξειδικευμένες γνώσεις και καθιστά επιτακτική την ανάγκη για τη δημιουργία υποστηρικτικών εργαλείων που θα διευκολύνουν την όλη διαδικασία ανάπτυξης.

4.2 Προκλήσεις δημιουργίας ενός περιβάλλοντος ανάπτυξης εφαρμογών

Μελετώντας τη βιβλιογραφία και κάποιες καλές κοινές πρακτικές από την ανάπτυξη επιτυχημένων συστημάτων λογισμικού διαπιστώνει κανείς πόσο σημαντική είναι η ύπαρξη υποστηρικτικών εργαλείων για τη βιωσιμότητα κάποιου συστήματος ή πλατφόρμας. Ειδικότερα, στο πεδίο του διάχυτου υπολογισμού παρατηρεί κανείς ότι ενώ πολύ συχνά κατασκευάζονται πάρα πολύ αξιόλογες πλατφόρμες, χωρίς την κατάλληλη υποστήριξη από εργαλεία ανάπτυξης εφαρμογών δεν κατορθώνουν τελικώς να υιοθετηθούν από τους τελικούς χρήστες και οι εφαρμογές πάνω από τις πλατφόρμες αυτές παραμένουν στο στάδιο των πρωτοτύπων. Αντίθετα, διάσημες πλατφόρμες με ολοένα αυξανόμενο αριθμό χρηστών συνεχίζουν να βρίσκονται στο προσκήνιο και να εξελίσσονται με γοργούς ρυθμούς. Για παράδειγμα, χαρακτηριστικό είναι το πλήθος των αποτελεσμάτων που παίρνει κανείς από μια απλή αναζήτηση στη μηχανή αναζήτησης της Google με λέξεις κλειδιά όπως «Sunspot Applications» και «TinyOS Applications». Τα αποτελέσματα αυτά διαμορφώνουν ξεκάθαρα την νέα τάση διείσδυσης αυτών των πλατφόρμων κάνοντας ακόμα πιο επιτακτική την ανάγκη για αποδοτικότερη ανάπτυξη και έλεγχο τέτοιου είδους εφαρμογών. Ωστόσο, οι ιδιαιτερότητες των πλατφόρμων που προορίζονται οι εφαρμογές αυτές καθώς και η χειροκίνητη ανάπτυξη τους απαιτεί υψηλό βαθμό εξειδίκευσης από τους προγραμματιστές. Η ύπαρξη υποστηρικτικών εργαλείων μπορεί να λύσει το πρόβλημα αυτό αυξάνοντας παράλληλα το επίπεδο αφαίρεσης στο οποίο θα αναπτύσσονται οι εφαρμογές αυτές. Ειδικοί του πεδίου

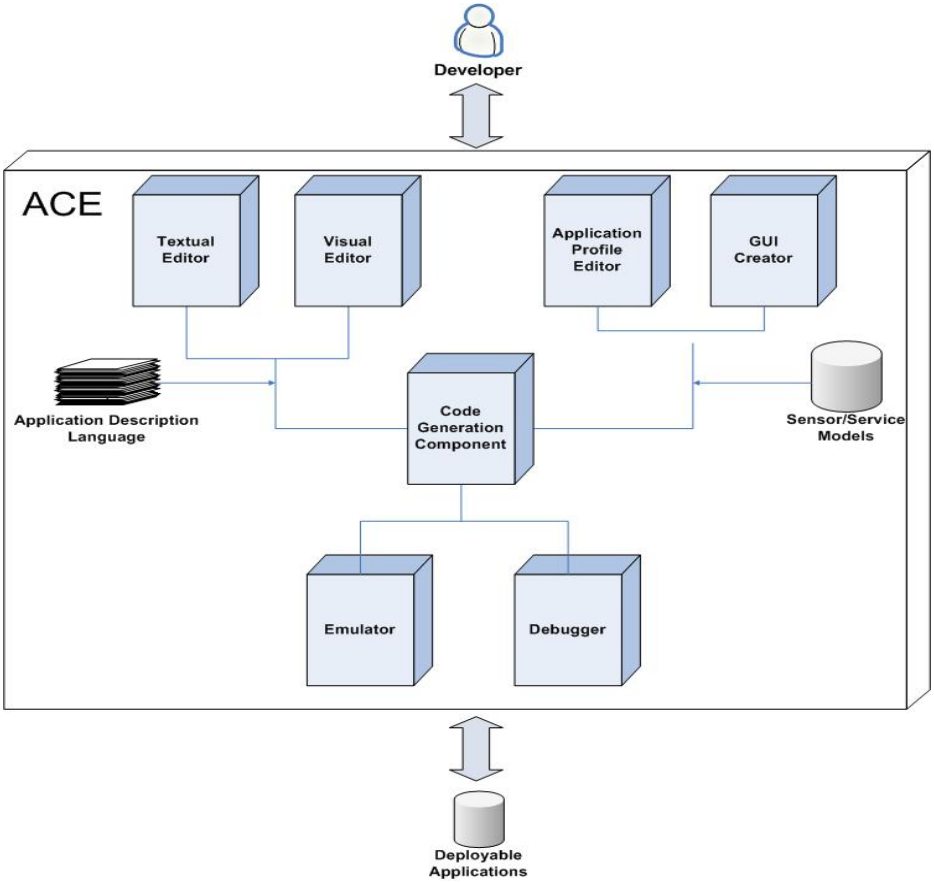
εφαρμογής και όχι έμπειροι προγραμματιστές θα πρέπει να είναι σε θέση να αναπτύξουν τέτοιου είδους εφαρμογές χωρίς αυτό να επηρεάζει την ποιότητα των παραγόμενων εφαρμογών. Αντιθέτως, η κατευθυνόμενη ανάπτυξη τέτοιων εφαρμογών θα πρέπει να οδηγεί σε παραγωγή υψηλής ποιότητας κώδικα που θα εκμεταλλεύεται καλές πρακτικές και σχεδιαστικά πρότυπα για την πλατφόρμα στόχο.

4.3 Η αρχιτεκτονική του συστήματος

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης και ελέγχου εφαρμογών αποτελείται από ένα σύνολο από εργαλεία τα οποία υποστηρίζουν το χρήστη σε ολόκληρη τη διαδικασία ανάπτυξης. Η κατασκευή ενός τέτοιου εύχρηστου περιβάλλοντος από την αρχή απαιτεί πάρα πολύ μεγάλη προσπάθεια και εμπειρία στον τομέα της ανάπτυξης υποστηρικτικών εργαλείων αλλά και άψογη γνώση του πεδίου για το οποίο προορίζεται. Συνήθως απαιτεί την ενασχόληση πολλών εξειδικευμένων προγραμματιστών που τις περισσότερες όμως φορές δεν είναι διαθέσιμοι στα πλαίσια μιας μικρής ομάδας ανάπτυξης. Τις περισσότερες φορές τέτοια περιβάλλοντα αναπτύσσονται από μεγάλους οίκους λογισμικού και είτε αποτελούν εμπορικά προϊόντα είτε υποστηρίζουν κάποια νέα πλατφόρμα ανάπτυξης. Αν και τα περισσότερα τέτοια περιβάλλοντα είναι κλειστού λογισμικού τα τελευταία χρόνια γίνεται μια προσπάθεια τα περιβάλλοντα αυτά να αποκτήσουν μια πιο αρθρωτή δομή όπου ο κάθε ενδιαφερόμενος θα μπορεί να αναπτύξει το δικό του περιφερειακό εργαλείο και να το ενσωματώσει στο ολοκληρωμένο περιβάλλον.

Στα πλαίσια της διπλωματικής εργασίας σχεδιάστηκε και υλοποιήθηκε ένα τέτοιο ολοκληρωμένο περιβάλλον ανάπτυξης και ελέγχου εφαρμογών για μια συγκεκριμένη πλατφόρμα διάχυτου υπολογισμού που αναπτύσσεται από την ερευνητική ομάδα διάχυτου υπολογισμού του τμήματος Πληροφορικής και Τηλεπικοινωνιών. Η προσέγγιση που ακολουθήθηκε ήταν να χρησιμοποιήσουμε την πλατφόρμα του Eclipse [47] και χρησιμοποιώντας το μηχανισμό των plug-ins που προσφέρει να επεκτείνουμε τις δυνατότητες του για υποστήριξη της δικής μας DSM λύσης. Αν και άλλα περιβάλλοντα ανάπτυξης όπως το NetBeans [48] παρέχουν πλέον παρόμοιους μηχανισμούς επέκτασης, η πλατφόρμα του Eclipse προτιμήθηκε εξαιτίας της ανοικτής της φύσης και της μεγάλης ενεργής κοινότητας χρηστών και προγραμματιστών που το υποστηρίζουν. Παράλληλα, ως η μοναδική αξιόλογη πλατφόρμα ανοικτού λογισμικού για την ανάπτυξη DSM λύσεων το Eclipse αναδείχθηκε ως μονόδρομος.

Για την ανάπτυξη εφαρμογών διάχυτου υπολογισμού σε ένα υψηλότερο επίπεδο αφαίρεσης αυτό που απαιτείται είναι η δημιουργία μιας νέας γλώσσας περιγραφής των εφαρμογών καθώς και ένα πλήθος από κατάλληλα υποστηρικτικά εργαλεία τα οποία θα ολοκληρώνονται σε ένα ενιαίο και εύχρηστο περιβάλλον. Η εικόνα 6 απεικονίζει την αρχιτεκτονική ενός τέτοιου ιδανικού περιβάλλοντος.



Εικόνα 6: Η γενική αρχιτεκτονική του συστήματος.

Με επίκεντρο τη γλώσσα μοντελοποίησης, κυρίαρχο ρόλο παίζουν τα εργαλεία επεξεργασίας των στιγμιότυπων της γλώσσας. Ανάλογα με τη διακριτή σύνταξη της γλώσσας τα εργαλεία αυτά μπορεί να κυμαίνονται από έναν απλό επεξεργαστή κειμένου έως και σύνθετους γραφικούς επεξεργαστές. Επιπλέον διάφορα περιφερειακά εργαλεία μπορούν να χρησιμοποιούνται για να δηλώσουν διάφορα χαρακτηριστικά μιας νέας εφαρμογής και τα οποία δεν καλύπτονται από τη γλώσσα. Για παράδειγμα, ένας επεξεργαστής του προφίλ μιας εφαρμογής (application profile editor) μπορεί να χρησιμοποιηθεί για την επεξεργασία δηλωτικών πληροφοριών που αφορούν την εφαρμογή αυτή και αποθηκεύονται ανεξάρτητα από την κύρια λογική της. Ωστόσο, στο επίκεντρο της διαδικασίας ανάπτυξης βρίσκεται ο γεννήτορας παραγωγής κώδικα, ο οποίος είναι υπεύθυνος για την μετατροπή της υψηλής επιπέδου περιγραφής της εφαρμογής σε εκτελέσιμο κώδικα κατάλληλο για εκτέλεση στην πλατφόρμα στόχο. Για

τον έλεγχο των εφαρμογών για εντοπισμό πιθανών λογικών λαθών και απρόβλεπτης συμπεριφοράς, εργαλεία ελέγχου κρίνονται απαραίτητα.

Ωστόσο, στα πλαίσια της διπλωματικής εξετάζουμε την ανάπτυξη των βασικότερων συστατικών της παραπάνω αρχιτεκτονικής για τη δημιουργία εφαρμογών διάχυτου υπολογισμού σε ένα υψηλότερο επίπεδο αφαίρεσης. Ορίζουμε μια νέα γλώσσα μοντελοποίησης περιγραφής εφαρμογών και αναπτύσσουμε τα βασικά εργαλεία επεξεργασίας των στιγμιότυπων της με σκοπό την παραγωγή εκτελέσιμων εφαρμογών για την πλατφόρμα στόχο.

4.3.1 Η γλώσσα περιγραφής εφαρμογών

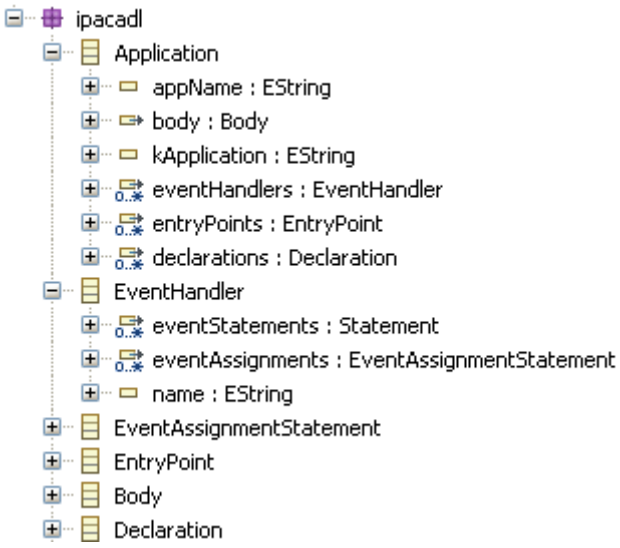
Η γλώσσα περιγραφής εφαρμογών (Application Description Language – ADL) [33] είναι μια γλώσσα ειδικού σκοπού και βασικά το πιο σημαντικό συστατικό του περιβάλλοντος ανάπτυξης εφαρμογών. Παρέχει όλα τα απαραίτητα στοιχεία και δομές που απαιτούνται για τον ορισμό μιας νέας εφαρμογής. Η σύνταξή της είναι απλή και συνδυάζει μερικά κοινά χαρακτηριστικά από πολύ γνωστές γλώσσες προγραμματισμού. Ακολουθεί μια προσέγγιση βασισμένη σε μοντέλα και το πλαίσιο Xtext [56] χρησιμοποιείται για τον ορισμό της γλώσσας και την παραγωγή των κατάλληλων εργαλείων.

Ο ορισμός της γλώσσας αποτελείται από ένα σύνολο 42 επεκταμένων EBNF κανόνων οι οποίοι χρησιμοποιούν ετικέτες (labeling) για τα μη τερματικά σύμβολα στο δεξιό μέρος του κανόνα της γραμματικής και αποτελούν τη βάση για την παραγωγή του κατάλληλου μεταμοντέλου. Κάθε κανόνας καθορίζει την σύνταξη του εκάστοτε στοιχείου της γλώσσας και αποτελείται από ένα συνδυασμό στατικών και δυναμικών κομματιών. Η εικόνα 7 απεικονίζει δύο βασικούς κανόνες από τον ορισμό της γλώσσας περιγραφής εφαρμογών ενώ μια αναλυτική παρουσίαση παρέχεται στο [33]. Το σύνολο των επεκταμένων EBNF κανόνων που ορίζουν την γλώσσα παρουσιάζεται επίσης στο παράρτημα Ι.

```
4  Application:
5      kApplication="application" appName=ID "{"
6          (declarations+=Declaration)*
7          (eventHandlers+=EventHandler)*
8          (entryPoints+=EntryPoint)*
9          body=Body
10     "}";
11
12 // Specifies the code to be executed for an entry or an event.
13 EventHandler:
14     "event" name=ID "{"
15         (eventAssignments+=EventAssignmentStatement)*
16         (eventStatements+=Statement)*
17     "}";
```

Εικόνα 7: Παράδειγμα κανόνων της γραμματικής που ορίζει τη γλώσσα.

Στην πράξη, οι κανόνες αυτοί χρησιμοποιούνται από το Xtext πλαίσιο και παράγουν την αφηρημένη σύνταξη της νέας γλώσσας υπό την μορφή μεταμοντέλου όπως ορίζει το Eclipse Modeling Framework. Η αντιστοίχιση μεταξύ των κανόνων της γραμματικής και των εννοιών του μεταμοντέλου ακολουθεί κάποια πολύ συγκεκριμένα πρότυπα. Η κεφαλή κάθε κανόνα αντιστοιχίζεται σε μια νέα κλάση (EClass) και κάθε γνώρισμα στο δεξιό μέρος του κανόνα αντιστοιχίζεται σε ένα γνώρισμα (EAttribute) της κλάσης. Οι αφηρημένοι κανόνες της γραμματικής μετατρέπονται επίσης σε αφηρημένες κλάσεις (abstract classes) στο μεταμοντέλο. Με τον τρόπο αυτό, το μεταμοντέλο περιγράφει τη δομή του αφηρημένου συντακτικού δέντρου (Abstract Syntax Tree – AST) και τη δομή των διακριτών μοντέλων που περιγράφονται σε όρους της γλώσσας περιγραφής εφαρμογών. Ένα στιγμιότυπο από το παραγόμενο μεταμοντέλο απεικονίζεται στην εικόνα 8. Για παράδειγμα, μια εφαρμογή (Application), αποτελείται από ένα όνομα (appName), το σώμα (body) της εφαρμογής (λογική της εφαρμογής) καθώς και από μηδέν ή περισσότερους χειριστές γεγονότων (eventHandlers), σημεία εισόδου εκτέλεσης (entryPoints) και δηλώσεις (declarations).



Εικόνα 8: Στιγμιότυπο από το μεταμοντέλο της γλώσσας περιγραφής εφαρμογών.

Παράλληλα, με τη χρήση του πλαισίου Xtext, παράγονται επίσης δυο επιπλέον βασικά συστατικά. Ένας parser για την επεξεργασία των μοντέλων βασισμένος στο πλαίσιο παραγωγής parser ANTLR [35], και ένας serializer ο οποίος παρέχει τη δυνατότητα σειριοποίησης των μοντέλων για μόνιμη αποθήκευση σε XMI μορφή.

Ας σημειωθεί ότι η αφηρημένη σύνταξη υπό μορφή μεταμοντέλου παρέχει όλα τα απαραίτητα μέσα για το στατικό και συντακτικό έλεγχο των μοντέλων που αναπτύσσονται με την βοήθεια της γλώσσας. Οι στατικοί έλεγχοι γίνονται κατά την διαδικασία ελέγχου του μοντέλου και ορίζονται ως περιορισμοί πάνω στην αφηρημένη σύνταξη του μεταμοντέλου. Ωστόσο, το πλαίσιο Xtext παρέχει και τον απαραίτητο μηχανισμό για τον εμπλουτισμό της σημασιολογίας του μεταμοντέλου, επιτρέποντας τη δυναμική επικύρωση των αναπτυσσόμενων μοντέλων. Οι έλεγχοι αυτοί υλοποιούνται με τη μορφή περιορισμών πάνω στο μεταμοντέλο ενώ οποτεδήποτε κάποιος περιορισμός παραβιάζεται, ένα μήνυμα λάθους εμφανίζεται στο χρήστη της γλώσσας. Το πλεονέκτημα αυτής της προσέγγισης είναι ότι όλοι οι περιορισμοί μπορούν να χρησιμοποιηθούν για όλες τις διακριτές συντάξεις που μπορούν να οριστούν για την αφηρημένη σύνταξη. Με τον τρόπο αυτό, ο έλεγχος του μοντέλου γίνεται μια κοινή λειτουργία και όχι μια λειτουργία που πρέπει να υλοποιηθεί για κάθε μία από τις διαφορετικές διακριτές συντάξεις.

Τέτοιου είδους περιορισμοί ορίζονται με την βοήθεια μια ειδικής γλώσσας ελέγχου (check language) υπό μορφή κανόνων. Η εικόνα 9 παρουσιάζει δυο παραδείγματα τέτοιων απλών κανόνων όπου ουσιαστικά δηλώνουν ότι στα πλαίσια της γλώσσας περιγραφής εφαρμογών που αναπτύξαμε, το όνομα μια εφαρμογής (appName) δεν

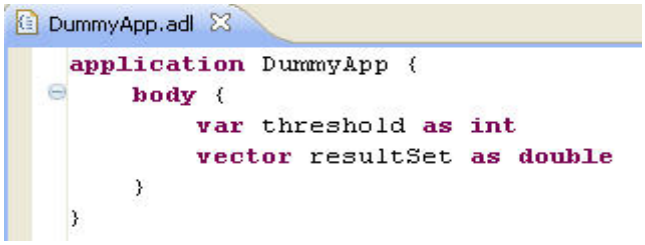
μπορεί να είναι κάποια δεσμευμένη λέξη του συντακτικού της γλώσσας, δεν μπορεί να ξεκινάει από ψηφίο και θα πρέπει να αποτελείται από τουλάχιστον δυο χαρακτήρες.

```
// The application name should not be a reserved word and it cannot start with a digit.
context Application ERROR
    "Invlaid application name!" :
    !isReserved(appName) && !startsWithDigit(appName);

// The application name should have length greater than 1.
context Application WARNING
    "The " + this.appName + " is too short." :
    this.appName.length > 1;
```

Εικόνα 9: Παραδείγματα απλών κανόνων που επεκτείνουν τη σημασιολογία του μεταμοντέλου της γλώσσας περιγραφής εφαρμογών.

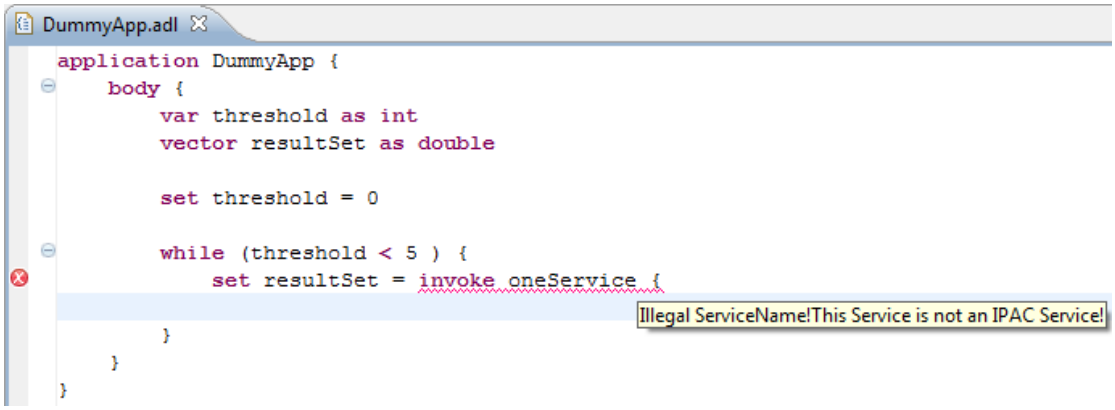
4.3.2 Ο επεξεργαστής κειμένου

Ένα άλλο θεμελιώδεις συστατικό του περιβάλλοντος ανάπτυξης είναι ο επεξεργαστής κειμένου (textual editor), ο οποίος παρέχει όλη την απαιτούμενη λειτουργικότητα για τον ορισμό νέων εφαρμογών. Υποστηρίζει πλούσιες δυνατότητες επεξεργασίας για την ανάπτυξη των εφαρμογών ενώ κληρονομεί όλους τους συνηθισμένους μηχανισμούς επεξεργασίας που παρέχει ένας επεξεργαστής στο περιβάλλον του Eclipse. Επισήμανση σύνταξης (syntax highlighting), βοήθεια περιεχομένου (content assist), παραγωγή block κώδικα (code snippets), αναδίπλωση κώδικα (code folding), outline όψη (outline view), αναζήτηση (searching) και μηχανισμό ελέγχου (checking mechanism) αποτελούν τις βασικές από τις παρεχόμενες ευκολίες. Χρώματα και γραμματοσειρές μπορούν να παραμετροποιηθούν ανάλογα με την σημασία κάθε στοιχείου της γλώσσας και σύμφωνα με τις προτιμήσεις του χρήστη. Η εικόνα 10 απεικονίζει ένα στιγμιότυπο του παραγόμενου επεξεργαστή κειμένου για την γλώσσα περιγραφής εφαρμογών. Οι δεσμευμένες λέξεις της γλώσσας επισημαίνονται με διαφορετικό χρώμα ενώ το σύμβολο μείον (-) στο αριστερό μέρος του επεξεργαστή υποδεικνύει την παρεχόμενη λειτουργικότητα της αναδίπλωσης κώδικα. Διαθέτοντας αυτή τη λειτουργία, block πηγαίου κώδικα σε όρους της γλώσσας περιγραφής εφαρμογών μπορούν να αναδιπλωθούν και να ξαναεμφανιστούν ανάλογα με την επιθυμία του χρήστη.



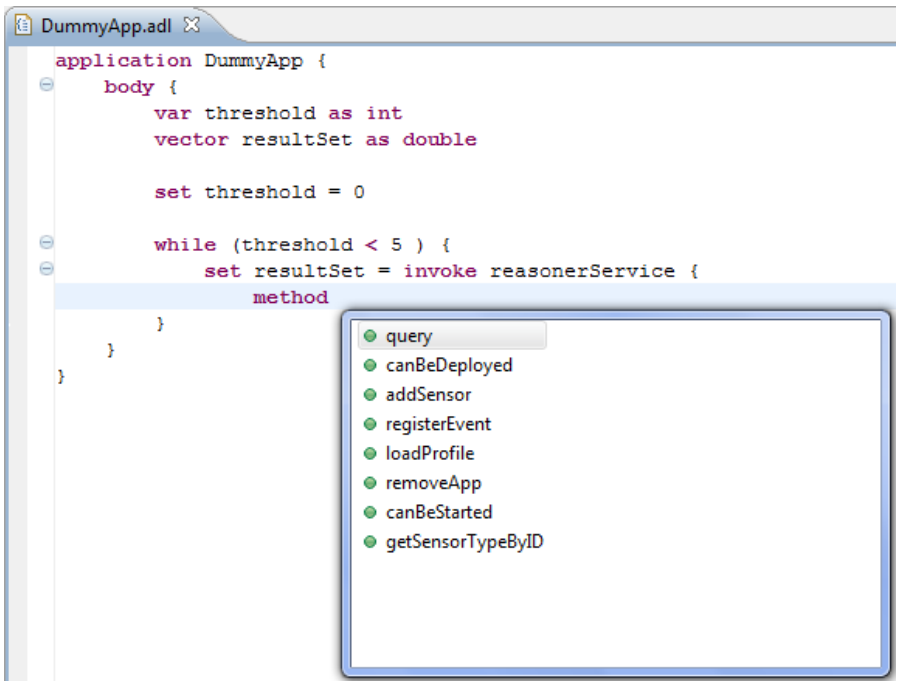
Εικόνα 10: Επισήμανση και αναδίπλωση κώδικα στον επεξεργαστή κειμένου.

Επιπρόσθετα, η εικόνα 11 παρουσιάζει ένα στιγμιότυπο από την διαδικασία ορισμού μιας εφαρμογής, με το μηχανισμό ελέγχου να ενεργοποιείται επειδή κάποιος περιορισμός στη σημασιολογία του μοντέλου παραβιάστηκε. Στην περίπτωση αυτή ο χρήστης ενημερώνεται με το κατάλληλο διαγνωστικό μήνυμα που έχει οριστεί ενώ στο αριστερό μέρος του επεξεργαστή το κατάλληλο σύμβολο υποδεικνύει τη γραμμή στην οποία εντοπίστηκε το λάθος.



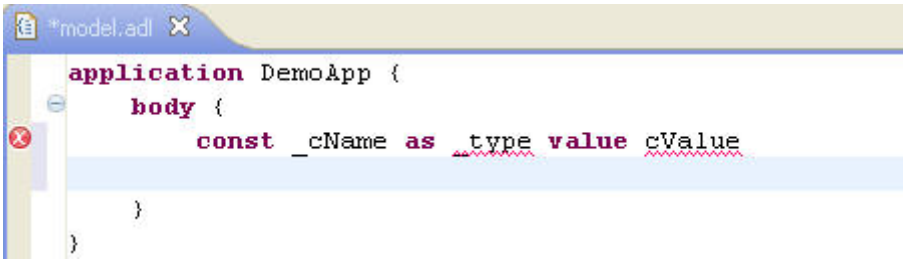
Εικόνα 11: Ο μηχανισμός ελέγχου στην πράξη.

Μια από τις περισσότερο χρήσιμες δυνατότητες του επεξεργαστή κειμένου είναι ο προηγμένος μηχανισμός βοήθειας περιεχομένου. Ο χρήστης μπορεί να εκμεταλλευτεί τον μηχανισμό αυτό ώστε να ορίσει νέες εφαρμογές με ένα περισσότερο αποδοτικό τρόπο. Χρησιμοποιώντας το συνδυασμό των πλήκτρων «Ctrl+Space» μια λίστα από όλα τα κατάλληλα στοιχεία της γλώσσας τα οποία μπορούν να χρησιμοποιηθούν σε αυτό το σημείο, προτείνονται και ο χρήστης μπορεί να επιλέξει το κατάλληλο. Ένα στιγμιότυπο λειτουργίας του μηχανισμού αυτού φαίνεται στην εικόνα 12 όπου προτείνονται στο χρήστη τα ονόματα των διαθέσιμων μεθόδων που μπορεί να χρησιμοποιήσει από την κλήση του συστατικού «reasonerService».



Εικόνα 12: Ο μηχανισμός βοήθειας περιεχομένου στην πράξη.

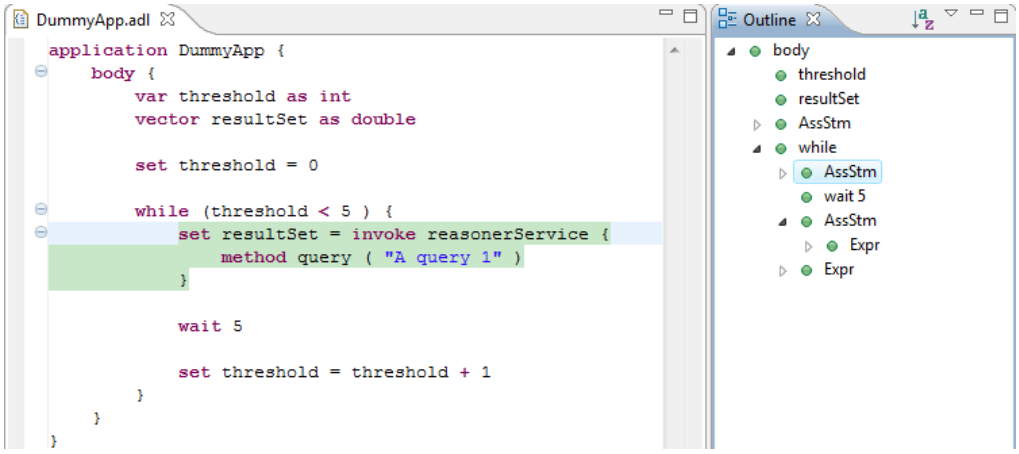
Άλλη μια ενδιαφέρουσα παρεχόμενη λειτουργία του επεξεργαστή είναι η δυναμική παραγωγή block κώδικα σε όρους της γλώσσας περιγραφής εφαρμογών. Ορίζοντας τους κατάλληλους κανόνες, ο χρήστης μπορεί να επιλέξει τα διαθέσιμα στοιχεία από τη λίστα με τις προτάσεις και τότε το κατάλληλο block κώδικα αυτόματα εισάγεται στον επεξεργαστή κειμένου. Στην εικόνα 13 ένα τέτοιο block κώδικα παρουσιάζεται για μια δήλωση σταθεράς. Οι επισημασμένες λέξεις αντιστοιχούν στις ανάλογες δεσμευμένες λέξεις της γλώσσας ενώ οι λέξεις που ξεκινούν με το σύμβολο «_» πρέπει να αντικατασταθούν κατάλληλα από το χρήστη.



Εικόνα 13: Αυτόματη παραγωγή block κώδικα για δήλωση σταθεράς σύμφωνα με τους όρους της γλώσσας περιγραφής εφαρμογών.

Ωστόσο, η outline όψη είναι μια επιπλέον πολύ ενδιαφέρουσα λειτουργία του επεξεργαστή κειμένου που κληρονομείται από την πλατφόρμα του Eclipse και απεικονίζει τη δομή του στιγμιότυπου του μοντέλου σε μια δενδρική μορφή. Η

δυνατότητα αυτή είναι πάρα πολύ χρήσιμη κατά τον ορισμό μεγάλων και πολύπλοκων εφαρμογών και σκοπός της είναι η γρήγορη πλοήγηση στον κώδικα της εφαρμογής. Όταν ο χρήστης επιλέξει συγκεκριμένα στοιχεία της εφαρμογής στην outline όψη, η επιλογή αυτή αυτόματα αντανακλάται στον κώδικα της εφαρμογής στον επεξεργαστή κειμένου και αντίστροφα. Ένα παράδειγμα της outline όψης παρουσιάζεται στην εικόνα 14.



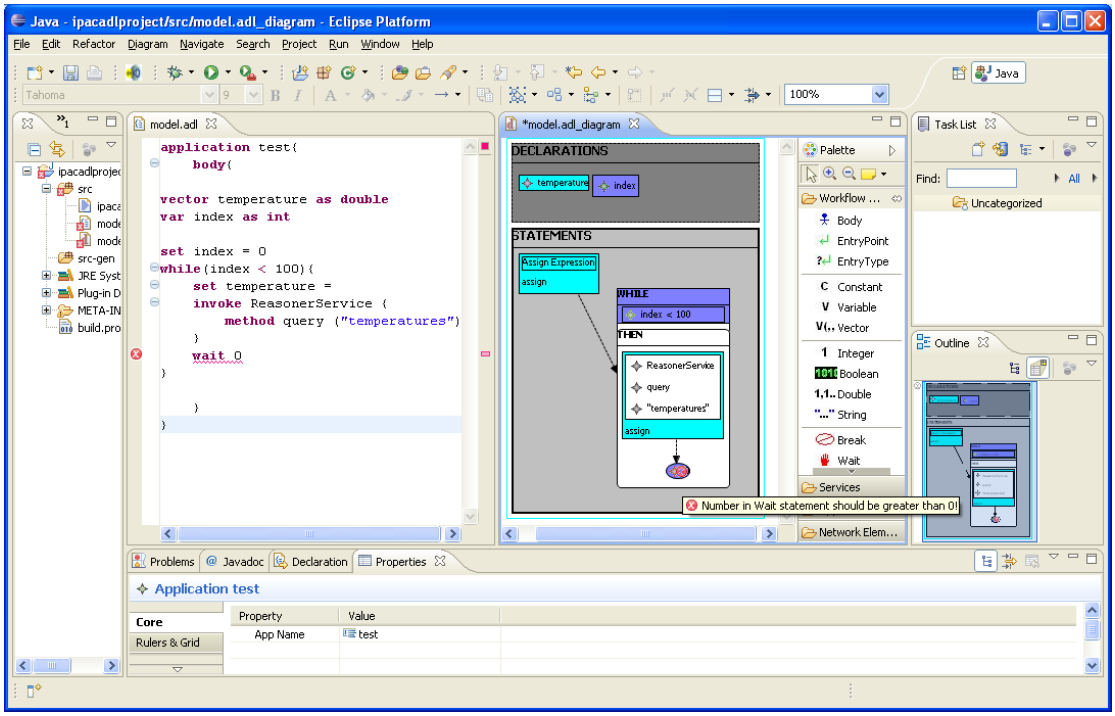
Εικόνα 14: Η λειτουργία της outline όψης στον επεξεργαστή κειμένου.

4.3.3 Ο γραφικός επεξεργαστής

Ο γραφικός επεξεργαστής (visual editor) είναι στενά συνδεδεμένος με το σχεδιασμό του επεξεργαστή κειμένου και υλοποιείται με την βοήθεια του Graphical Modeling Framework. Χρησιμοποιώντας ένα σχεδιασμό ροής (workflow-design), αποτελείται από ένα σύνολο από εργαλειοθήκες και εργαλεία τα οποία ένας χρήστης χρειάζεται ώστε να αναπτύξει εφαρμογές, ακολουθώντας drag-and-drop παράδειγμα. Κάθε βασικό στοιχείο της γλώσσας περιγραφής εφαρμογών διαθέτει την κατάλληλη γραφική αναπαράσταση και μέσω της πλευρικής εργαλειοθήκης του γραφικού επεξεργαστή γίνεται διαθέσιμο στο χρήστη. Το κύριος μέρος του επεξεργαστή αποτελείται από τον καμβά πάνω στον οποίο αναπτύσσεται η εκάστοτε εφαρμογή. Για την αποτελεσματικότερη τοποθέτηση των στοιχείων πάνω στον καμβά κατάλληλοι layout διαχειριστές μπορούν να ενεργοποιηθούν ανάλογα με τις προτιμήσεις του χρήστη. Όπως και στον επεξεργαστή κειμένου έτσι και στο γραφικό επεξεργαστή η ανάλογη outline όψη διευκολύνει την πλοήγηση στο κώδικα της εφαρμογής.

Επίσης, ένας μηχανισμός ελέγχου είναι διαθέσιμος και αναγνωρίζει ασυνέπειες στη λογική (business logic) της εφαρμογής κατά την διάρκεια της δημιουργίας. Ας σημειωθεί

ότι και οι δύο επεξεργαστές είναι αμφίδρομα συγχρονισμένοι. Αλλαγές και ασυνέπειες στον κώδικα της εφαρμογής αντικατοπτρίζονται ταυτόχρονα και στους δύο επεξεργαστές. Στην πράξη, ο χρήστης επεξεργάζεται τον κώδικα της εφαρμογής σε έναν από τους δύο επεξεργαστές και στο παρασκήνιο το κατάλληλο μοντέλο δημιουργείται. Ταυτόχρονα, ελέγχονται οι συντακτικοί και σημασιολογικοί περιορισμοί που έχουν οριστεί πάνω στο μεταμοντέλο της γλώσσας περιγραφής εφαρμογών και η διαδικασία αυτή συνεχώς επαναλαμβάνεται δίνοντας στο χρήστη την αίσθηση ότι επεξεργάζεται το μοντέλο απ' ευθείας. Με τον τρόπο αυτό ο χρήστης ανά πάσα στιγμή μπορεί να χρησιμοποιήσει οποιοδήποτε από τους δύο επεξεργαστές επιθυμεί και να παράξει έγκυρα στιγμιότυπα στοιχείων της γλώσσας χωρίς καμία διαφορά στο παραγόμενο αποτέλεσμα. Ας σημειωθεί, ότι λόγο της φύσης του, ο επεξεργαστής κειμένου μπορεί να χρησιμοποιηθεί από έμπειρους χρήστες για την υλοποίηση πιο σύνθετων εφαρμογών οι οποίες δύσκολα θα μπορούσαν να υλοποιηθούν γραφικά.



Εικόνα 15: Αμφίδρομος συγχρονισμός μεταξύ των δύο επεξεργαστών. Ο μηχανισμός ελέγχου στην πράξη.

4.3.4 Ο γεννήτορας κώδικα

Ο γεννήτορας κώδικα (code generator) είναι ένα πολύ σημαντικό συστατικό του περιβάλλοντος ανάπτυξης. Αν και η λειτουργία του είναι διαφανής στο χρήστη, διασυνδέεται με το γραφικό επεξεργαστή και τον επεξεργαστή κειμένου μέσω προκαθορισμένων προτύπων (templates). Χρησιμοποιώντας τη σημασιολογία των

στοιχείων της γλώσσας μοντελοποίησης παράγει το τελικό Java κώδικα της εφαρμογής. Η είσοδος του γεννήτορα είναι ο ορισμός της λογικής της εφαρμογής σε όρους της γλώσσας περιγραφής εφαρμογών και ως έξοδο παράγεται ο πραγματικός κώδικας της εφαρμογής έτοιμος για να εγκατασταθεί στο ενδιάμεσο λογισμικό που υπάρχει διαθέσιμο στους κόμβους.

4.3.4.1 Η γλώσσα προτύπων

Η λειτουργία του γεννήτορα κώδικα βασίζεται στη γλώσσα προτύπων XPand [53]. Η βασική ιδέα πίσω από αυτό το συστατικό είναι η χρήση προκαθορισμένων προτύπων με στατικά και δυναμικά κομμάτια. Τα στατικά κομμάτια αποτελούνται από hard-coded κώδικα ενώ τα δυναμικά παράγονται αυτόματα. Λαμβάνοντας υπόψη τη σημασιολογία κάθε στοιχείου της γλώσσας περιγραφής εφαρμογών ο γεννήτορας χρησιμοποιεί ή παράγει την κατάλληλη έξοδο. Όπως περιγράφηκε προηγουμένως, όταν ο χρήστης χρησιμοποιεί κάποιον από τους δύο διαθέσιμους επεξεργαστές (για επεξεργασία κειμένου ή γραφικό) ένα μοντέλο δημιουργείται στο παρασκήνιο σύμφωνα με τον ορισμό και τους περιορισμούς του μεταμοντέλου της γλώσσας. Το μοντέλο αυτό κατασκευάζεται δυναμικά και περιέχει όλη την απαραίτητη πληροφορία για τα στοιχεία της γλώσσας που χρησιμοποιούνται στον ορισμό της εκάστοτε εφαρμογής. Παρέχοντας αυτή την πληροφορία στο γεννήτορα, η όλη διαδικασία είναι η ανάλυση και η επεξεργασία του μοντέλου και η παραγωγή του τελικού κώδικα. Επιπλέον, το προφίλ της εφαρμογής με τον ορισμό των δηλωτικών κομματιών και συγκεκριμένων γεγονότων (events) που ορίζονται για αυτήν, χρησιμοποιείται από το γεννήτορα. Ο γεννήτορας αναλύει τις πληροφορίες που βρίσκονται στο προφίλ και αντίστοιχα παράγει τον κατάλληλο κώδικα για το μηχανισμό διαχείρισης των events (event handling mechanism).

Ωστόσο, η έξοδος του γεννήτορα είναι ο πραγματικός κώδικας της εφαρμογής έτοιμος να εγκατασταθεί στο ενδιάμεσο λογισμικό των κινητών κόμβων. Χρησιμοποιώντας ως πλατφόρμα το περιβάλλον του OSGi, η εφαρμογή πρέπει να πακεταριστεί σε ένα OSGi bundle και άρα ένα σύνολο από εξωτερικά αρχεία πρέπει να δημιουργηθούν. Για παράδειγμα, η εικόνα 16 απεικονίζει στιγμιότυπο από ένα πρότυπο που παράγει την «Activator» κλάση της εφαρμογής όπως ορίζεται από την OSGi πλατφόρμα. Τα στατικά κομμάτια του παραγόμενου κώδικα επισημαίνονται με μπλε χρώμα ενώ τα άλλα αντιστοιχούν στα δυναμικά.

```
30public class Activator implements BundleActivator {
31
32    // Application specific information.
33    public static final String prefix = "«getAppEventPrefix()»";
34
35    «IF hasEvents(this)»
36    public static EventAdmin ea;
37    public ServiceTracker eaTracker;
38    «ENDIF»
39
40    «IF usesUIService()»
41    public ServiceTracker uiTracker;
42    public static eu.ipac.ui.osgi.Activator uis = null;
43    «ENDIF»
44
45    // Application Declaration.
46    public «firstUpperCase(splitAppName(this.appName))» app;
47
48    // Event Handler Declaration.
49    «IF hasEvents()»
50        «FOREACH this.eventHandlers AS e ITERATOR iter-»
51            «firstUpperCase(splitAppName(this.appName))».«firstUpperCase(e.name)»EventHandler
52        «ENDFOREACH-»
53    «ENDIF-»
54
55    public void start(BundleContext context) throws Exception {
56        // TODO: Remove after testing.
57        System.out.println("IPAC Application started!!!");
58
59        app = new «firstUpperCase(splitAppName(this.appName))»();
```

Εικόνα 16: Στιγμιότυπο από το πρότυπο για την παραγωγή της «Activator» κλάσης του bundle της εφαρμογής.

4.3.4.2 Η μηχανή εκτέλεσης ροής εργασιών

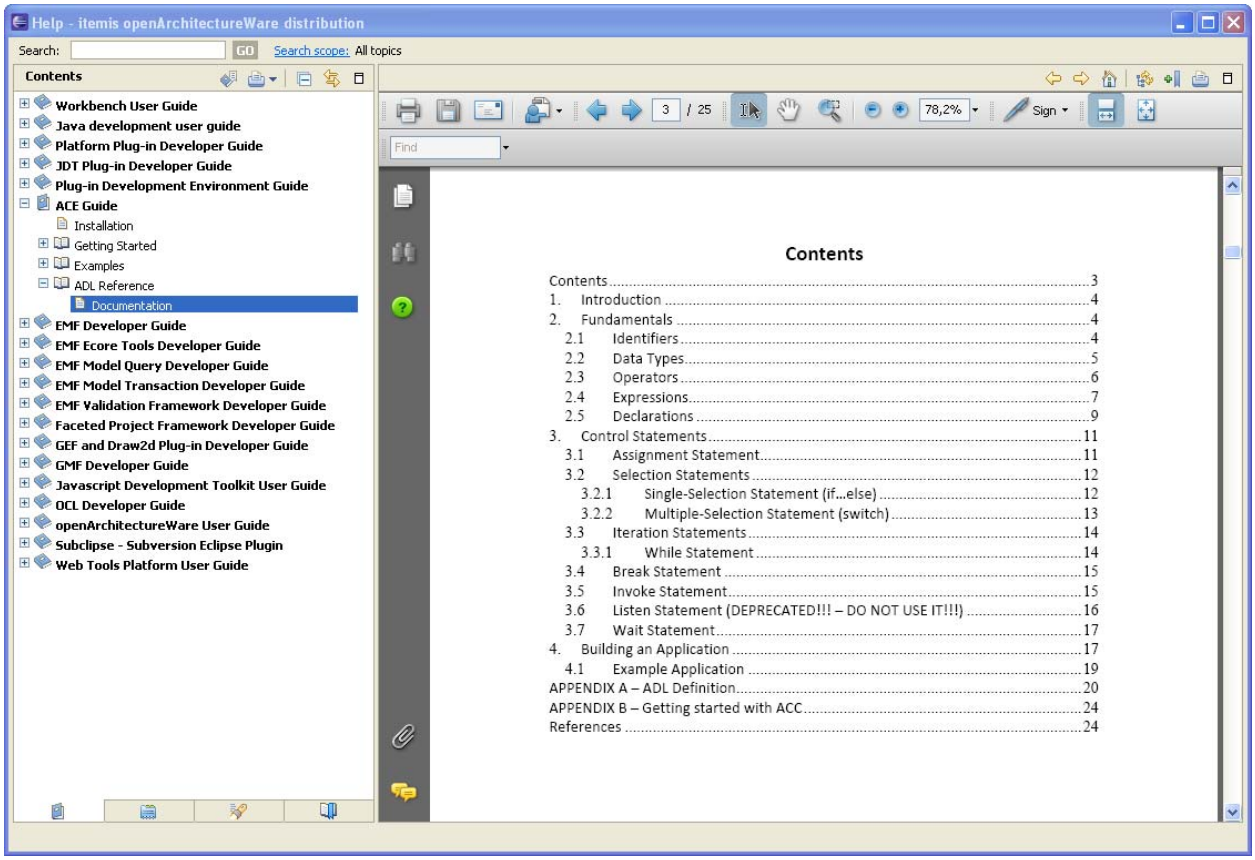
Η μηχανή εκτέλεσης ροής εργασιών (Model Workflow Engine - MWE) [54] είναι το συστατικό του περιβάλλοντος ανάπτυξης που αναλαμβάνει τη λογική οργάνωση ολόκληρης της διαδικασίας ανάπτυξης μιας εφαρμογής. Είναι το συστατικό εκείνο που ενεργοποιεί την τελική παραγωγή κώδικα και για το σκοπό αυτό χρησιμοποιεί κάποιο αρχείο ρυθμίσεων με δομή παρόμοια με Ant XML αρχεία [64]. Η όλη διαδικασία ξεκινά με το parsing του στιγμιότυπου του μοντέλου σε όρους της γλώσσας περιγραφής εφαρμογών, ακολουθεί ο έλεγχος της ορθότητας του και τελικά η παραγωγή της εφαρμογής στην επιθυμητή μορφή. Η εικόνα 17 παρουσιάζει το κυριότερο κομμάτι της λογικής του αρχείου ρυθμίσεων. Ας σημειωθεί ότι σε οποιοδήποτε κομμάτι της εξόδου πρέπει να παραχθεί Java κώδικας, αυτός ακολουθεί προκαθορισμένες προδιαγραφές μορφοποίησης με ενεργοποίηση του κατάλληλου «JavaBeautifier» συστατικού που χρησιμοποιείται.

```
10  <!-- Parse the model instance. -->
11  <component file='eu/ipac/ace/adl/parser/Parser.oaw'>
12     <modelFile value='${modelFile}'/>
13     <outputSlot value='theModel'/>
14  </component>
15
16  <!-- Clean the target directory. -->
17  <component class='oaw.workflow.common.DirectoryCleaner' directory='${targetDir}'/>
18  <component class='oaw.workflow.common.DirectoryCleaner' directory='${distDir}'/>
19
20  <!-- Serialize the model instance in XMI format. -->
21  <component class="oaw.emf.XmiWriter">
22     <inputSlot value='theModel'/>
23     <modelFile value="${targetDir}/${modelFile}.xmi"/>
24  </component>
25
26  <!-- Generate the source code. Use the code generation templates.
27       The generated code is Javabeautified. -->
28  <component class='oaw.xpand2.Generator'>
29     <metaModel id='mm' class='org.eclipse.m2t.type.emf.EmfRegistryMetaModel'/>
30     <expand value='eu::ipac::ace::adl::templates::Main::main FOR theModel'/>
31     <genPath value='${targetDir}'/>
32     <beautifier class='oaw.xpand2.output.JavaBeautifier' />
33  </component>
34
35</workflow>
```

Εικόνα 17: Στιγμιότυπο από το αρχείο ρυθμίσεων της μηχανής εκτέλεσης ροής εργασιών.

4.3.4.3 Ο μηχανισμός βοήθειας

Υιοθετώντας την αρχή ότι «τα εργαλεία πρέπει να συνοδεύονται από καλή τεκμηρίωση (documentation), το περιβάλλον περιλαμβάνει και ένα πάρα πολύ ισχυρό και εύχρηστο μηχανισμό βοήθειας ο οποίος μπορεί να ανακληθεί ανά πάσα στιγμή από το κεντρικό μενού του περιβάλλοντος. Στην πράξη, ο μηχανισμός αυτός επεκτείνει το μηχανισμό βοήθειας του Eclipse και εκμεταλλεύεται πλήρως την λειτουργικότητα που αυτός παρέχει. Ο χρήστης πολύ εύκολα πλέον μπορεί να αναζητήσει πληροφορίες για κάποιο συστατικό του περιβάλλοντος καθώς και να βρει τον πλήρη οδηγό της γλώσσας περιγραφής εφαρμογών. Η εικόνα 18 απεικονίζει ένα στιγμιότυπο του μηχανισμού βοήθειας στη πράξη.



Εικόνα 18: Ο μηχανισμός βοήθειας στην πράξη. Προεπισκόπηση του οδηγού της γλώσσας περιγραφής εφαρμογών με χρήση του ενσωματωμένου pdf viewer.

4.3.5 Το πλαίσιο εκτέλεσης

Όπως έχει αναφερθεί και παραπάνω, οι εφαρμογές που παράγονται από το περιβάλλον ανάπτυξης εφαρμογών στοχεύουν στην εγκατάσταση στο ενδιάμεσο λογισμικό που υπάρχει διαθέσιμο στους κόμβους. Το ενδιάμεσο αυτό λογισμικό βασίζεται πάνω στην πλατφόρμα του OSGi και χρησιμοποιεί μια αρχιτεκτονική βασισμένη σε συστατικά (component based). Το ενδιάμεσο αυτό στρώμα διαχωρίζει το επίπεδο των εφαρμογών από την πλατφόρμα στόχο. Αποτελείται από built-in συστατικά της πλατφόρμας του OSGi, όπως ο μηχανισμός διαχείρισης των events, όπως και custom συστατικά τα οποία έχουν αναπτυχθεί για το σκοπό αυτό (συστατικό επικοινωνίας, συστατικό διαχείρισης αισθητήρων κ.α). Οι εφαρμογές που παράγονται από το περιβάλλον ανάπτυξης τελικά κάνουν απλή χρήση των παραπάνω συστατικών αποκρύπτοντας έτσι την πολυπλοκότητα της πλατφόρμας και αυξάνοντας το επίπεδο της αφαίρεσης στο οποίο αυτές δημιουργούνται.

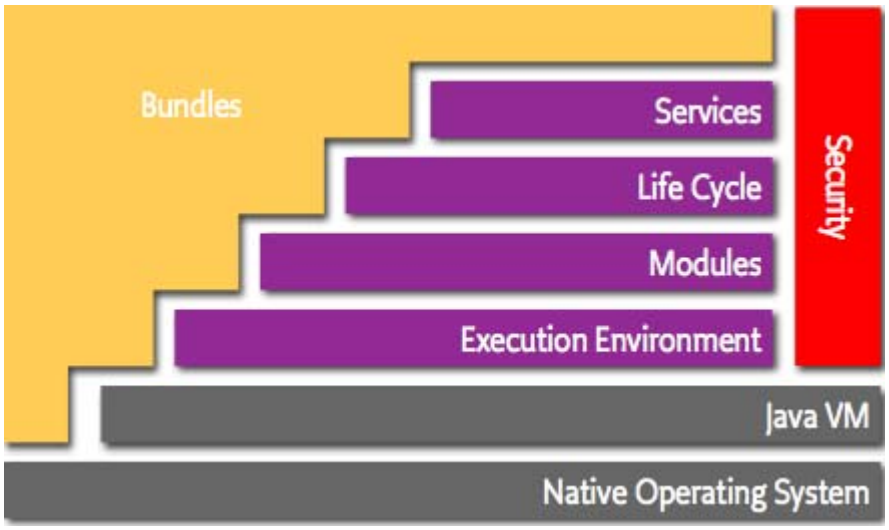
4.3.6 Η πλατφόρμα στόχος

Πλατφόρμα στόχος των αναπτυσσόμενων εφαρμογών καθορίστηκε οποιαδήποτε συσκευή υποστηρίζει την εκτέλεση Java εφαρμογών στη πλατφόρμα του OSGi. Η συσκευή θα πρέπει επίσης να διαθέτει δυνατότητες ασύρματης επικοινωνίας ενώ μπορεί να υποστηρίζει ένα πλήθος από συγκεκριμένους τύπους αισθητήρων οι οποίοι υποστηρίζονται από την πλατφόρμα την οποία αναπτύσσει η ομάδα διάχυτου υπολογισμού.

5. ΤΕΧΝΟΛΟΓΙΕΣ ΥΛΟΠΟΙΗΣΗΣ

5.1 Open Service Gateway initiative (OSGi)

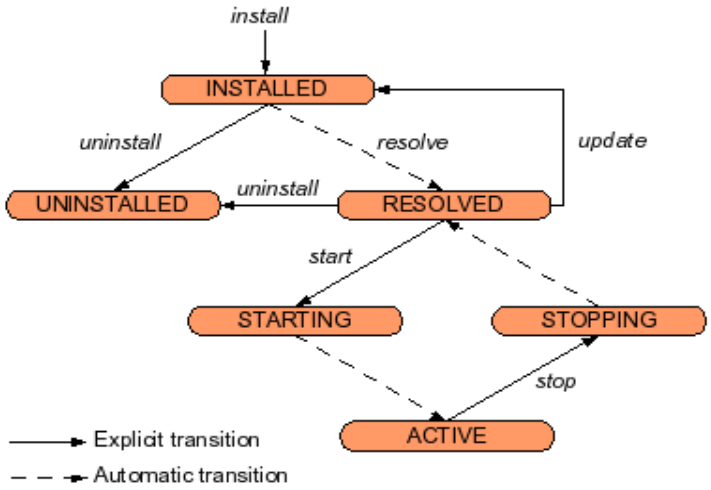
Η τεχνολογία του OSGi (Open Service gateway initiative) [42, 43] είναι ένα σύνολο από προδιαγραφές οι οποίες ορίζουν ένα δυναμικό σύστημα συστατικών (dynamic component system) για τη γλώσσα Java. Οι προδιαγραφές αυτές καθιστούν δυνατό ένα μοντέλο ανάπτυξης στο οποίο οι εφαρμογές συντίθενται δυναμικά από αρκετά διαφορετικά επαναχρησιμοποιήσιμα (reusable) συστατικά. Με τον τρόπο αυτό, τα συστατικά κρύβουν τις λεπτομέρειες υλοποίησης τους από άλλα αντίστοιχα συστατικά καθώς επικοινωνούν αποκλειστικά μέσω συγκεκριμένων υπηρεσιών (services) και μηχανισμών επέκτασης που αυτά ορίζουν. Ωστόσο, το κύριο μέρος των προδιαγραφών της OSGi τεχνολογίας, είναι το OSGi πλαίσιο (OSGi framework) το οποίο ορίζει ένα μοντέλο διαχείρισης του κύκλου ζωής μιας εφαρμογής, μια πλατφόρμα διαχείρισης υπηρεσιών και ένα περιβάλλον εγκατάστασης και εκτέλεσης των εφαρμογών. Η εικόνα 19 απεικονίζει συνοπτικά την στρωματοποιημένη αρχιτεκτονική του OSGi πλαισίου.



Εικόνα 19: Η στρωματοποιημένη αρχιτεκτονική του OSGi πλαισίου.

Τα bundles είναι τα βασικά OSGi συστατικά, τα οποία αναπτύσσονται από τους προγραμματιστές και εμπερικλείουν όλη την λογική ενός συστατικού/υπηρεσίας και τα οποία πακετάρονται στην κλασική μορφή ενός Java Archive (JAR) αρχείου της Java. Στη γενική περίπτωση, κάθε bundle είναι μια επεκτάσιμη μονάδα η οποία σε συνδυασμό με άλλα bundles μπορεί να δημιουργήσει καλά καθορισμένες σχέσεις εξάρτησης κατά τη ανάπτυξη ενός πολύπλοκου συστήματος. Τα bundles μπορούν να εγκατασταθούν (install), να εκκινήσουν, να σταματήσουν, να ενημερωθούν (update) και

να απεγκατασταθούν (unistall) χωρίς να απαιτηθεί να σταματήσει η λειτουργία άλλων bundles. Το επίπεδο υπηρεσιών συνδέει τα bundles με δυναμικό τρόπο προσφέροντας ένα μοντέλο δημοσίευσης-εξερεύνηση-προσαρμογής (publish-find-bind model) το οποίο τους επιτρέπει να εντοπίσουν την προσθήκη νέων υπηρεσιών ή την αφαίρεση τους και να προσαρμοστούν ανάλογα. Ο τυπικός κύκλος ζωής ενός OSGi bundle απεικονίζεται στην εικόνα 20.



Εικόνα 20: Ο κύκλος ζωής ενός bundle.

Ας σημειωθεί ότι οι προδιαγραφές του OSGi διευθύνονται από το OSGi Alliance [42], ένα forum το οποίο αποτελείται από αρκετές πολύ γνωστές εταιρείες και οργανισμούς. Αρχικός σκοπός της προσπάθειας αυτής ήταν κυρίως η ανάπτυξη εφαρμογών για δικτυακές συσκευές, περιβάλλοντα κινητού υπολογισμού, εφαρμογές τηλεματικής κ.α. Τα τελευταία χρόνια όμως, αρκετές OSGi αξιόλογες διανομές είναι πλέον διαθέσιμες για εγκατάσταση και μπορούν να χρησιμοποιηθούν σε πάρα πολλά διαφορετικά πεδία ενδιαφέροντος.

5.2 Η πλατφόρμα του Eclipse

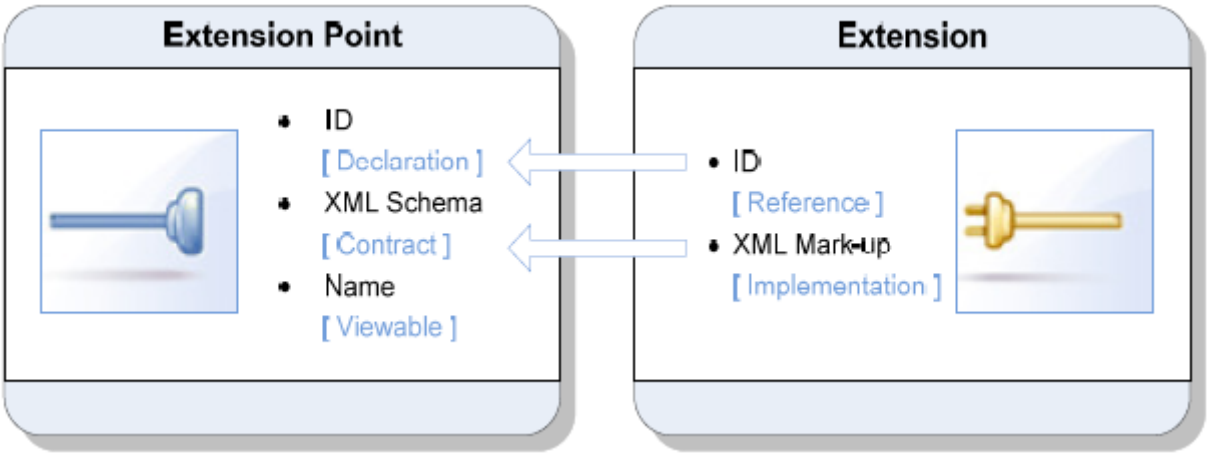
Η πλατφόρμα του Eclipse [32, 47] είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) που περιλαμβάνει την απαιτούμενη λειτουργικότητα για την κατασκευή προσαρμοσμένων ολοκληρωμένων περιβαλλόντων ανάπτυξης για άλλες γλώσσες προγραμματισμού. Είναι μια σύνθεση από συστατικά όπου χρησιμοποιώντας ένα υποσύνολο από αυτά, μπορεί κάποιος να χτίσει εργαλεία και εφαρμογές. Η περισσότερη από την λειτουργικότητα του είναι γενική καθώς η πλατφόρμα είναι κατασκευασμένη πάνω σε ένα μηχανισμό για ανακάλυψη, επέκταση, ολοκλήρωση και εκτέλεση συστατικών τα οποία ονομάζονται plug-ins.

5.2.1 Η αρχιτεκτονική των plug-ins

Τα plug-ins αποτελούν τη μικρότερη μονάδα λειτουργικότητας στην πλατφόρμα του Eclipse. Η ίδια η πλατφόρμα, η γραφική διασύνδεση του χρήστη (GUI) και τα εργαλεία που την επεκτείνουν αποτελούνται από plug-ins. Ένα απλό εργαλείο μπορεί να αποτελείται μόνο από ένα plug-in αλλά περισσότερο πολύπλοκα εργαλεία διαχωρίζονται τυπικά σε πολύ περισσότερα. Κάθε plug-in παρέχει την απαιτούμενη λειτουργικότητα που μπορεί να χρησιμοποιηθεί από το χρήστη ή να επαναχρησιμοποιηθεί και να επεκταθεί από άλλα plug-ins.

Έτσι, στην αρχιτεκτονική του Eclipse δύο είναι τα βασικά συστατικά: η πλατφόρμα εκτέλεσης (platform runtime) και ένα σύνολο από plug-ins. Η πλατφόρμα εκτέλεσης στηρίζεται πάνω στην πλατφόρμα υπηρεσιών του OSGi και ουσιαστικά είναι ένας μικρός OSGi πυρήνας (microkernel) υπεύθυνος για τη διαχείριση του κύκλου ζωής των plug-ins. Για το λόγο αυτό, ο OSGi όρος «bundle» που περιγράφει τη βασική μονάδα λειτουργικότητας πολλές φορές χρησιμοποιείται εναλλακτικά με τον όρο plug-in.

Βέβαια, στην πράξη, ένα plug-in δεν είναι τίποτε άλλο από ένα Java Archive (JAR) αρχείο που περιλαμβάνει όλους τους απαραίτητους πόρους για την εκτέλεση του, όπως Java κώδικα, εικόνες, κείμενο και ότι άλλο απαιτείται. Επίσης περιλαμβάνει δύο manifest αρχεία. Το βασισμένο σε OSGi manifest αρχείο, ονομαζόμενο «META-INF/MANIFEST.MF», περιγράφει βασικές πληροφορίες για το plug-in (πεδία «Bundle-Name», «Bundle-Version κ.α) και παρέχει μεταξύ των άλλων την απαραίτητη πληροφορία για τις εξαρτήσεις του plug-in με άλλα («Require-Bundle», «Export-Package»). Επίσης, το plug-in manifest αρχείο («plugin.xml») δηλώνει ρητά τις διασυνδέσεις ενός plug-in με άλλα plug-ins ορίζοντας τα λεγόμενα σημεία επέκτασης (extension points) που αυτό προσφέρει και τις πιθανές επεκτάσεις (extensions) που αυτό χρησιμοποιεί/υλοποιεί. Κάθε σημείο επέκτασης καθορίζεται μοναδικά από ένα αναγνωριστικό που διαθέτει ενώ η περιγραφή της κατάλληλης διασύνδεσης υλοποιείται με χρήση του κατάλληλου XML σχήματος (εικόνα 21). Παραστατικά, θα μπορούσε κανείς να φανταστεί ένα σημείο επέκτασης σα μια ηλεκτρική πρίζα (electric outlet) και μια επέκταση σαν την κατάλληλη πρίζα (plug) που ταιριάζει σε αυτή.



Εικόνα 21: Διαισθητική αναπαράσταση του μηχανισμού διασύνδεσης των plug-ins.

5.2.2 Η γραφική πλατφόρμα διασύνδεσης χρήστη

Η γραφική πλατφόρμα διασύνδεσης χρήστη του Eclipse (Platform UI) βασίζεται πάνω σε ένα βασικό επεκτάσιμο περιβάλλον εργασίας (workbench) το οποίο υποστηρίζει όλες τις απαραίτητες δομές για τη δημιουργία του κατάλληλου ολοκληρωμένου περιβάλλοντος αλληλεπίδρασης με το χρήστη. Αποτελείται κυρίως από επεξεργαστές (editors) και όψεις (views) ενώ η κατάλληλη διευθέτηση όλων αυτών στο κύριο περιβάλλον εργασίας συνθέτει τα ανάλογα perspectives. Όταν ένας επεξεργαστής ή μία όψη είναι ενεργά, μπορούν να συνεισφέρουν τις ανάλογες ενέργειες (actions) στα βασικά κύρια μενού (menus) του περιβάλλοντος και στις εργαλειοθήκες (toolbars).

Ωστόσο, γραφικές επεκτάσεις στο κύριο περιβάλλον του Eclipse μπορούν εύκολα να αναπτυχθούν χρησιμοποιώντας της εργαλειοθήκες Standard Widget Toolkit (SWT) και JFace [40]. Η SWT [41] είναι μια ανοιχτού λογισμικού γραφική βιβλιοθήκη για τη Java, η οποία παρέχει ένα σύνολο από ανεξάρτητα λειτουργικού συστήματος (OS-independent) widgets, σχεδιασμένη να παρέχει μια αποτελεσματική και φορητή λύση για ανάπτυξη γραφικών διεπαφών χρήστη. Παράλληλα, η JFace είναι μια υψηλότερου επιπέδου εργαλειοθήκη, η οποία χρησιμοποιεί την SWT και την επεκτείνει παρέχοντας αποτελεσματικούς μηχανισμούς για την διαχείριση γραφικών στοιχείων όπως λίστες, δέντρα, πίνακες ενώ προσθέτει και το κατάλληλο πλαίσιο διαχείρισης ενεργειών (action framework).

5.3 Το Xtext πλαίσιο

Το Xtext [56] είναι ένα πλαίσιο για την γρήγορη ανάπτυξη γλωσσών ειδικού σκοπού και άλλων γλωσσών προγραμματισμού υπό μορφή κειμένου και την ενσωμάτωσή τους στο περιβάλλον του Eclipse. Συνδέεται στενά με το Eclipse Modeling Framework (EMF) και επεκτείνει τις δυνατότητες της πλατφόρμας του Eclipse για την ανάπτυξη ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης για γλώσσες ειδικού σκοπού. Σε αντίθεση με τα συνήθη εργαλεία παραγωγής parser (π.χ JavaCC ή ANTLR), το Xtext παρέχει πολλά περισσότερα από την απλή παραγωγή ενός συντακτικού (parser) και ένα λεκτικού αναλυτή (lexical analyzer) δεδομένης μια γραμματικής. Στην πράξη, η γραμματική περιγράφει της έγκυρες δομές της εκάστοτε γλώσσας και αποτελεί τη βάση πάνω στην οποία παράγεται ένα σύνολο από συστατικά. Αναλυτικότερα, το πλαίσιο χρησιμοποιείται για να παράξει α) ένα συντακτικό και λεκτικό αναλυτή βασισμένο στο ANTLR για την ανάγνωση των μοντέλων από μορφή κειμένου, β) μια αντιστοίχιση της γραμματικής σε μεταμοντέλο και των στιγμιότυπων αυτής σε Ecore μοντέλα, γ) ένα serializer για την αποθήκευση των μοντέλων σε μορφή κειμένου, δ) έναν linker για την εγκαθίδρυση συνδέσεων μεταξύ των στοιχείων του μοντέλου (model elements) και ε) όλους τους κατάλληλους μηχανισμούς για την ενσωμάτωση της νέας γλώσσας στο περιβάλλον του Eclipse.

Στα πλαίσια της διπλωματικής το πλαίσιο Xtext χρησιμοποιήθηκε γα τον ορισμό της γλώσσας περιγραφής εφαρμογών και για την ενσωμάτωση της στο περιβάλλον του Eclipse. Κάθε επιπλέον συστατικό του περιβάλλοντος ανάπτυξης εφαρμογών υλοποιήθηκε ως ξεχωριστό plug-in [4] δημιουργώντας έτσι τις προδιαγραφές για τη δημιουργία ενός επεκτάσιμου ολοκληρωμένου περιβάλλοντος ανάπτυξης και ελέγχου εφαρμογών για το διάχυτο υπολογισμό. Συγκεκριμένα, για την ανάπτυξη των plug-ins χρησιμοποιήθηκε το Plug-in Development Environment (PDE) [58] του Eclipse ενώ μια λίστα αποτελούμενη από τα κυριότερα plug-ins που αναπτύχθηκαν περιλαμβάνει τα:

- *«eu.ipac.ace.adl»:* ορίζει την γλώσσα περιγραφής εφαρμογών. Ο πλήρης ορισμός της γλώσσας βρίσκεται στο αρχείο “*IpacADL.xtxt*”.
- *«eu.ipac.ace.adl.editor»*: υλοποιεί τον επεξεργαστή κειμένου για τα στιγμιότυπα της γλώσσας περιγραφής εφαρμογών. Αρχεία ιδιαίτερου ενδιαφέροντος είναι τα ”*ContentAssist.ext*” και “*EditorExtensions.ext*”.

- *«eu.ipac.ace.gmf.editor / eu.ipac.ace.gmf.diagram / eu.ipac.ace.gmf.edit / eu.ipac.ace.gmf»*: υλοποιούν τον γραφικό επεξεργαστή για τα στιγμιότυπα της γλώσσας περιγραφής εφαρμογών.
- *«eu.ipac.ace.adl.generator»*: υλοποιεί τον γεννήτορα παραγωγής κώδικα.
- *«eu.ipac.ace.help»*: υλοποιεί τον μηχανισμό βοηθείας του περιβάλλοντος ανάπτυξης εφαρμογών.
- *«eu.ipac.ace.ui»*: υλοποιεί τα λοιπά γραφικά στοιχεία που επεκτείνουν την default διεπαφή του Eclipse (wizards, dialogs).
- *«eu.ipac.ace.resources»*: περιλαμβάνει όλες τις απαραίτητες εξωτερικές βιβλιοθήκες που απαιτούνται για την ανάπτυξη των υπόλοιπων plug-ins

Όσον αφορά την πλατφόρμα πάνω στην οποία στοχεύουν οι παραγόμενες εφαρμογές αυτή στηρίζεται πάνω στην υποκείμενο πλαίσιο του OSGi και συγκεκριμένα σε συσκευές οι οποίες υποστηρίζουν την εκτέλεση Java εφαρμογών μέσα στο πλαίσιο αυτό.

6. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

6.1 Σύνοψη και συμπεράσματα

Στην παρούσα εργασία μελετήθηκαν τεχνικές με τις οποίες μπορούμε να αναπτύξουμε γλώσσες ειδικού σκοπού καθώς και υποστηρικτικά εργαλεία για αυτές. Οι παραδοσιακές τεχνικές ανάπτυξης γλωσσών και εργαλείων αν και καλά θεμελιωμένες και για χρόνια χρησιμοποιήσιμες στην πράξη, φαίνεται να παρουσιάζουν κάποια σημαντικά μειονεκτήματα. Οι απαιτήσεις των σύγχρονων γλωσσών έχουν πλέον αυξηθεί και η κλασική μορφή των γλωσσών υπό μορφή κειμένου φαίνεται να μην είναι αρκετή σε πολλές περιπτώσεις. Συχνά μια γλώσσα απαιτεί περισσότερες από μια διακριτές αναπαραστάσεις ή περιορίζεται μόνο σε μια γραφική αναπαράσταση απαιτώντας βέβαια και τα ανάλογα εργαλεία επεξεργασίας. Εναλλακτικές περιγραφές γλωσσών απαιτούνται οι οποίες θα υποβοηθούν με αυτόματο ή ημι-αυτόματο τρόπο την ανάπτυξη και των κατάλληλων υποστηρικτικών εργαλείων. Τα τελευταία χρόνια, η προσέγγιση βασισμένη σε μοντέλα υπόσχεται λύσεις στο παραπάνω πρόβλημα και φαίνεται να το αντιμετωπίζει με αρκετά ενθαρρυντικά αποτελέσματα.

Ωστόσο, η ανάπτυξη βασισμένη σε μοντέλα δεν είναι μια καινοτόμα ιδέα. Οι αλλαγές που σφυρηλατούνται από αυτή τη προσέγγιση ίσως φαίνονται ριζοσπαστικές, αλλά κατά βάθος υπάρχουν τρεις απλές πρακτικές τις οποίες οποιοσδήποτε έμπειρος μηχανικός λογισμικού αναγνωρίζει: α) μην επαναλαμβάνεσαι, β) αυτοματοποίησε μετά από τρεις εμφανίσεις και γ) προσαρμοσμένες λύσεις ταιριάζουν καλύτερα από τις γενικές. Έτσι, πάρα πολλές από τις αρχές και τεχνικές που χρησιμοποιούνται έχουν αναπτυχθεί και παλαιότερα σε διάφορα πεδία. Η δημιουργία γραφικών γλωσσών, τα component frameworks είναι έννοιες γνωστές που έχουν χρησιμοποιηθεί και παλαιότερα αλλά από ένα ίσως μικρότερο αριθμό από ανθρώπους. Επίσης, η αυτόματη παραγωγή ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης εφαρμογών από μια περιγραφή γλώσσας δεν είναι κάτι καινούριο. Στην πράξη ένας αριθμός από metacase εργαλεία υπάρχει που προσεγγίζει αυτή την ιδέα ενώ η κοινότητα των γραφικών γραμματικών έχει επίσης κάνει αρκετή δουλειά στην παραγωγή τέτοιων περιβαλλόντων [26], [27]. Αυτό όμως που δίνει μεγάλη ώθηση σε αυτή τη «νέα» προσέγγιση ανάπτυξης λογισμικού είναι το γεγονός ότι ένας αριθμός από υπάρχουσες τεχνικές συγκεντρώνεται πλέον σε ένα συνεκτικό σύνολο.

Μελετώντας λοιπόν τις βασικές αρχές της ανάπτυξης βασισμένης σε μοντέλα επιχειρήσαμε την σχεδίαση και ανάπτυξη ενός ολοκληρωμένου περιβάλλοντος

ανάπτυξης εφαρμογών για μια συγκεκριμένη πλατφόρμα διάχυτου υπολογισμού. Περιγράψαμε την αρχιτεκτονική ενός τέτοιου περιβάλλοντος και υλοποιήσαμε τα βασικά συστατικά του στηριζόμενοι στις αρχές αυτές. Το περιβάλλον που αναπτύχθηκε παρέχει τελικά στους χρήστες μια πληθώρα από ευκολίες για τον ορισμό νέων εφαρμογών, ενώ χαρακτηριστικά του όπως ο συγχρονισμός μεταξύ του επεξεργαστή κειμένου και του γραφικού επεξεργαστή το καθιστούν ως ένα πολύ αξιόλογο περιβάλλον ανάπτυξης. Βέβαια, όπως σε όλε τις περιπτώσεις παραγωγής λογισμικού, η εξ' ολοκλήρου δημιουργία ενός ολοκληρωμένου περιβάλλοντος με το χέρι θα μπορούσε να παράξει ένα καλύτερο και πιο αποτελεσματικό περιβάλλον. Παρόλα αυτά, είναι σημαντικό να συγκρίνει κανείς τον χρόνο και την προσπάθεια που απαιτούνται για να κατασκευαστεί ένα αξιόλογο περιβάλλον χρησιμοποιώντας τη πλατφόρμα του Eclipse με το χρόνο και την προσπάθεια που απαιτείται για τη δημιουργία ενός τέλειου παρόμοιου περιβάλλοντος χειροκίνητα. Ειδικότερα τώρα, στο πεδίο του διάχυτου και κινητού υπολογισμού που οι εξελίξεις είναι ραγδαίες και η φύση των εφαρμογών παρουσιάζει κάποια πολύ ιδιαίτερα χαρακτηριστικά, η γρήγορη και ποιοτική παραγωγή γλωσσών και εργαλείων κρίνεται αναγκαία.

Στο σημείο αυτό, πρέπει να γίνει επίσης κατανοητό ότι η ιδέα της ανάπτυξης βασισμένης σε μοντέλα, σε καμία περίπτωση δεν είναι η αντικατάσταση των προγραμματιστών από τους γεννήτορες, αλλά το να καταστήσουν τους ίδιους τους προγραμματιστές να παράγουν τα ίδια συστήματα γρηγορότερα, πιο αξιόπιστα και με μεγαλύτερη ποιότητα. Ένα γεννήτορας ο οποίος ορίζεται από έναν ειδικό, χωρίς αμφιβολία, παράγει εφαρμογές γρηγορότερα και με καλύτερη ποιότητα από αυτές που παράγονται χειροκίνητα από μέτριους προγραμματιστές. Αυτό εξάλλου είναι και το πρόβλημα των περισσότερων έργων λογισμικού τα οποία καθυστερούν είτε αποτυγχάνουν και περιλαμβάνουν πάρα πολλά bugs. Ωστόσο, η εισαγωγή των DSM εννοιών και εργαλείων στις ομάδες ανάπτυξης (development teams) είναι μια διαδικασία η οποία συνεπάγεται μια αλλαγή παραδείγματος (paradigm shift).

Επίσης, συχνά λέγεται ότι τα διάφορα πλαίσια (frameworks) και τα APIs είναι το ίδιο όπως οι γλώσσες ειδικού σκοπού, επιτρέποντας στους προγραμματιστές να δουλεύουν με ένα διαφορετικό σύνολο από έννοιες, συχνά σε ένα υψηλότερο επίπεδο αφαίρεσης. Ωστόσο η βασική διαφορά έγκειται στο γεγονός ότι με τη χρήση ενός API το εκάστοτε αποτέλεσμα γράφεται και πάλι στη γλώσσα προγραμματισμού που ο προγραμματιστής χρησιμοποιούσε παλαιότερα. Αντιθέτως, μια εξωτερική γλώσσα ειδικού σκοπού

προσφέρει σε έναν χρήστη όχι μόνο πιο αφηρημένες έννοιες αλλά επίσης μια νέα σύνταξη.

Τέλος, ένα χρήσιμο μάθημα που αποκτήθηκε μέσα από αυτή τη διπλωματική εργασία είναι ότι σε αντίθεση με παλαιότερα δεν είναι πια ανάγκη να στηρίζεται κάποιος αποκλειστικά σε κάποιον κατασκευαστή εργαλείων για να αναπτύξει εργαλεία για μια συγκεκριμένη πλατφόρμα. Η τεχνολογία πλέον υπάρχει, συνεχώς εξελίσσεται, είναι ανοικτή και απλά απαιτείται η προσαρμογή στο εκάστοτε πεδίο ενδιαφέροντος.

6.2 Μελλοντική εργασία

Σε καμία περίπτωση η διπλωματική αυτή εργασία δεν εξάντλησε όλα τα θέματα τα οποία σχετίζονται με την ανάπτυξη βασισμένη σε μοντέλα, γεγονός που αφήνει περιθώρια για περαιτέρω μελέτη και επεκτάσεις. Θέματα όπως η εξέλιξη των μεταμοντέλων και η αυτόματη αναπροσαρμογή των στιγμιότυπων τους αποτελούν ένα πολύ ενδιαφέρον πεδίο έρευνας το οποίο παρουσιάζει πολλά κοινά στοιχεία με τεχνικές δανεισμένες από άλλα ερευνητικά πεδία όπως π.χ οι βάσεις δεδομένων (schema evolution).

Τα μεταμοντέλα εξελίσσονται με το χρόνο και απαιτούνται καλά καθορισμένα βήματα ώστε το νέο μοντέλο που προκύπτει να είναι σύμφωνο με κάποιες θεμελιώδης αρχές. Αρκετές ερευνητικές προσπάθειες οδεύουν προς αυτή την κατεύθυνση με τους συγγραφείς του [24] να περιγράφουν την εξέλιξη των μεταμοντέλων ως ένα σύνολο μετασχηματισμών. Βέβαια και τα μοντέλα πρέπει να συνεξελίσσονται ώστε να παραμένουν συμβατά με το μεταμοντέλο. Χωρίς συνεξέλιξη, τα μοντέλα παύουν να είναι έγκυρα. Με τα τωρινά δεδομένα, σε ένα τυπικό σενάριο, η εξέλιξη των μεταμοντέλων και η συνεξέλιξη των μοντέλων πραγματοποιείται χειροκίνητα. Αυτή είναι μια διαδικασία αρκετά επιρρεπής σε λάθη, που μπορεί εύκολα να οδηγήσει σε ασυνέπειες μεταξύ του μεταμοντέλου και των σχετιζόμενων μοντέλων. Απαιτείται λοιπόν ένας αυτόματος τρόπος από βήματα συνεξέλιξης που συμπεραίνονται από πολύ καλά καθορισμένα βήματα εξέλιξης. Στο σημείο αυτό κάποιος θα μπορούσε να μελετήσει τεχνικές που έχουν προταθεί για «αναδιαμόρφωση αντικειμενοστραφούς κώδικα» (refactoring object-oriented code) [21], [22] ως και τεχνικές για προσαρμογή γραμματικών (grammar adaptation) από το ευρύτερο πεδίο της μηχανικής γραμματικών [23]. Ωστόσο, τα βασικά θέματα παραμένουν η διατήρηση της σημασιολογίας και η διατήρηση των στιγμιότυπων. Επίσης, ένα σημείο με ιδιαίτερη σημασία είναι η συνεξέλιξη του συνόλου των υποστηρικτικών εργαλείων (κειμενογράφοι, γεννήτορες κ.α) που χρησιμοποιούνται

για την επεξεργασία των μοντέλων και οι οποίοι θα πρέπει να παράγονται με έναν ημιαυτόματο τρόπο.

Όσον αφορά το ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών που σχεδιάστηκε και υλοποιήθηκε, η αρθρωτή του αρχιτεκτονική αφήνει περιθώρια για πάρα πολλές επεκτάσεις και δημιουργία πολλών περιφερειακών εργαλείων. Λαμβάνοντας υπόψη τις αυξανόμενες δυνατότητες των κόμβων στους οποίους μπορούν να εγκατασταθούν οι παραγόμενες εφαρμογές αλλά και τους περιορισμούς που αυτοί θέτουν, εργαλεία όπως ο δημιουργός μιας μινιμαλιστικής γραφικής διεπαφής (minimalistic GUI Creator) μπορούν να αναπτυχθούν. Υψίστης σημασίας είναι και η υλοποίηση εργαλείων ελέγχου για τις παραγόμενες εφαρμογές πριν αυτές εγκατασταθούν στους κόμβους. Όπως σχεδόν κάθε γλώσσα γενικού σκοπού έχει το δικό της αποσφαλματωτή (debugger), ιδανικά και ο χρήστης θα πρέπει να είναι ικανός να κάνει αποσφαλμάτωση στο επίπεδο της γλώσσας περιγραφής εφαρμογών και όχι στον παραγόμενο κώδικα Java. Για το σκοπό αυτό ένας μηχανισμός αντιστοίχισης μεταξύ των δομών της γλώσσας και του παραγόμενου κώδικα θα πρέπει να μελετηθεί και να αναπτυχθεί. Προς την κατεύθυνση αυτή, τεχνικές αντίστροφης μηχανικής (reverse engineering) καθώς και χρήση του DSL Debugger Framework (DDF) [30] βασισμένο στο περιβάλλον του Eclipse μπορούν να μελετηθούν. Τέλος, η δημιουργία ενός εργαλείου προσομοίωσης των εφαρμογών πριν αυτές εγκατασταθούν στους πραγματικούς κόμβους θα μπορούσε να βοηθήσει στην ανίχνευση σφαλμάτων σε πρώιμα στάδια.

ΠΑΡΑΡΤΗΜΑ Ι – ΟΡΙΣΜΟΣ ΓΛΩΣΣΑΣ ΠΕΡΙΓΡΑΦΗΣ ΕΦΑΡΜΟΓΩΝ

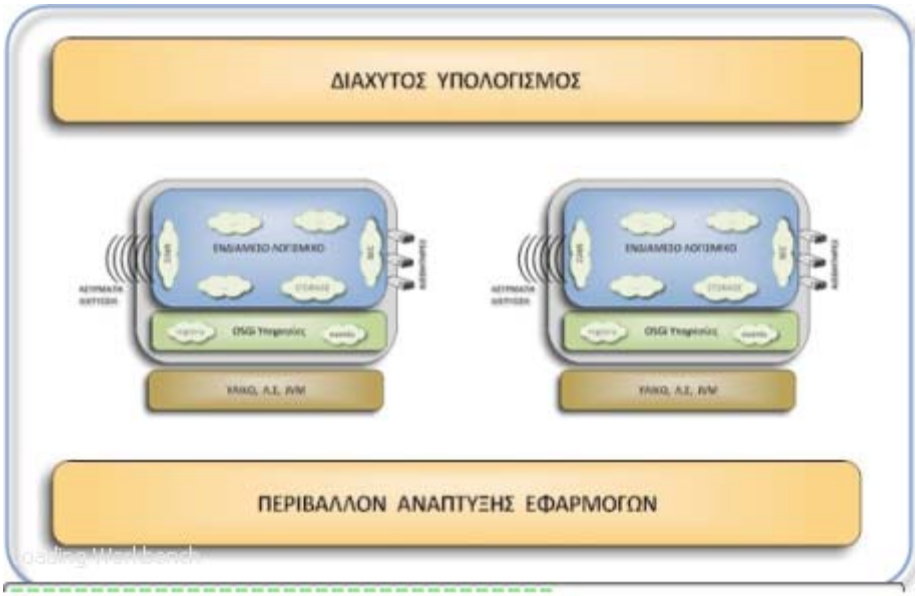
```
Application:
      kApplication="application" appName=ID "{"
            (declarations+=Declaration)*
            (eventHandlers+=EventHandler)*
          (entryPoints+=EntryPoint)*
            body=Body
      "}";

EventHandler:
      "event" name=ID "{"
            (eventAssignments+=EventAssignmentStatement)*
            (eventStatements+=Statement)*
      "}";

EventAssignmentStatement:
      "property" assRef=[VariableDeclaration] "value"
handlerRef=[EventHandler] "." propId=ID;

EntryPoint:
      "entry" name=ID "{"
            (entryStatements+=Statement)*
      "}";

Body:
      "body" "{"
            (statements+=Statement)*
      "}";

// ********************** //
// ***** DECLARATIONS ***** //
// ********************** //
// Abstract rule. Useful for linking references!
Declaration:
      ConstantDeclaration | VariableDeclaration | VectorDeclaration;

ConstantDeclaration:
      "const" name=ID "as" type=TypeName "value" cValue=ConstantValue;

VariableDeclaration :
      "var" name=ID "as" type=TypeName ("value" cValue=ConstantValue)?;

VectorDeclaration :
      "vector" name=ID "as" type=TypeName;

Enum TypeName :
      int="int" | double="double" | string="String" | boolean="boolean";

// ***** CONSTANT VALUES ***** //
ConstantValue:
       IntegerLiteral | StringLiteral | DoubleLiteral | BooleanLiteral;

IntegerLiteral:
      intValue=INT;

StringLiteral:
      strValue=STRING;

BooleanLiteral:
      booleanValue=BooleanEnum;

Enum BooleanEnum:
```

```
      true="true" | false="false";

DoubleLiteral:
      doubleValue=DOUBLE;

// A native rule is a lexer rule!
Native DOUBLE:
      "('0'..'9')*'.'('0'..'9')+";

// ********************* //
// ***** STATEMENTS ***** //
// ********************* //
Statement:
      AssignmentStatement | IfStatement | SwitchStatement | InvokeStatement |
ListenStatement | WaitStatement | IterationStatement | BreakStatement;

IfStatement :
           "if" "(" ifExpression=Expression ")" "{"
                 (ifStatements+=Statement)*
            "}"
           ("else" "{"
                 (elseStatements+=Statement)*
            "}")?;

SwitchStatement :
           "switch" "(" switchExpression=Expression ")" "{"
                 (caseStatements+=CaseStatement)*
                 (defaultStatement=DefaultStatement)?
                 "}";

// ATTENTION: Only INT values in CaseStatements!
CaseStatement:
      "case" cValue=INT ":" (caseStatements+=Statement)*;

DefaultStatement:
      "default:"  (defaultStatements+=Statement)*;

ListenStatement :
      "listen" eventName=[EventHandler] eventType=EventType;

Enum EventType :
      blocked="blocked" | nonBlocked="nonBlocked";

// ATTENTION: Java style! Space seperator at the parameter list!
InvokeStatement:
      "invoke" componentName=ID "{"
                  "method" methodName=ID
                        primitiveEL=PrimitiveElementList
                        (faultStatement=FaultStatement)?
      "}";

PrimitiveElementList :
      "(" (primitiveElements+=PrimitiveElement ","
primitiveElements+=PrimitiveElement)*)?  ")"
;

// Like try-catch block. The code to be executed.
FaultStatement :
      "onFault" "{" (fStatements+=Statement)* "}";

// ATTENTION: A Declaration reference may be a constant declaration. Check is
needed!!!
// Also a PrimitiveElement is allowed as right operand. Checks needed
(possible?) or sth more simplified.
```

```
AssignmentStatement:
       "set" assRef=[Declaration] "=" (invokeStatement=InvokeStatement |
expression=Expression);

IterationStatement:
      "while" "(" whileExpression=Expression ")" "{"
            (whileStatements+=Statement)*
      "}";

BreakStatement:
      "break";

WaitStatement:
      "wait" secs=INT;

// ***************************** //
// ***** PRIMITIVE ELEMENTS ***** //
// ***************************** //
// ATTENTION: Usage of the abstract rule! Vector reference is allowed!
PrimitiveElement:
       peRef=[Declaration] | cValue=ConstantValue;

// ********************** //
// ***** EXPRESSIONS ***** //
// ********************** //
Expression:
      condAndExpr=ConditionalAndExpression ("or"
optCondAndExpr+=ConditionalAndExpression)*;

ConditionalAndExpression:
      equalExpr=EqualityExpression ("and" optEqualExpr+=EqualityExpression)*;

EqualityExpression:
      relExpr=RelationalExpression (equalOps+=EqualityOperator
optRelExprs+=RelationalExpression)?;

RelationalExpression:
      addExpr=AdditiveExpression (relOps+=RelationalOperator
optAddExprs+=AdditiveExpression)?;

AdditiveExpression:
      multiExpr=MultiplicativeExpression (addOps+=AdditiveOperator
optMultiExprs+=MultiplicativeExpression)*;

// ATTENTION: PrmitiveElement or sth else more limited?
MultiplicativeExpression:
      primElem=PrimitiveElement (multOps+=MultiplicativeOperator
optPrimElems+=PrimitiveElement)*;

Enum EqualityOperator:
      equalOp="==" | unEqualOp="!=";

Enum RelationalOperator:
       leOp="<=" | geOp=">=" | lOp="<" | gOp=">";

Enum AdditiveOperator:
      addOp="+" | minusOp="-";

Enum MultiplicativeOperator:
      multOp="*" | divOp="/" | modOp="%";
```

ΠΑΡΑΡΤΗΜΑ ΙΙ - ΛΕΠΤΟΜΕΡΕΙΕΣ ΕΚΤΕΛΕΣΗΣ

Για την εκτέλεση του περιβάλλοντος ανάπτυξης απαιτείται η χρήση του περιβάλλοντος «Eclipse 3.4.2 (Ganymede) + openArchitectureWare 4.3.1» που υπάρχει διαθέσιμη στο http://oaw.itemis.com/openarchitectureware/language=en/2837/downloads καθώς και το σύνολο των plug-ins που αναπτύχθηκαν στα πλαίσια της διπλωματικής εργασίας. Για την εγκατάσταση των plug-ins αρκεί η αντιγραφή τους από τον κατάλογο «ΥΛΟΠΟΙΗΣΗ/PLUGINS» του συνοδευτικού ψηφιακού δίσκου στο κατάλογο plug-ins που βρίσκεται μέσα στον κατάλογο εγκατάστασης του Eclipse. Η εικόνα 22 απεικονίζει την εικόνα εκκίνησης του περιβάλλοντος ανάπτυξης εφαρμογών η οποία πρέπει να εμφανιστεί στην περίπτωση που η εγκατάσταση έχει ολοκληρωθεί επιτυχώς.



Εικόνα 22: Αρχική οθόνη εκκίνησης του περιβάλλοντος ανάπτυξης εφαρμογών.

Για περισσότερες λεπτομέρειες σχετικά με τη παρεχόμενη λειτουργικότητα και τον τρόπο χρήσης του περιβάλλοντος ανάπτυξης, ο χρήστης μπορεί να ανατρέξει στο μηχανισμό βοήθειας (παρ. 4.3.4.3) ο οποίος είναι διαθέσιμος από το κεντρικό μενού μέσω των επιλογών «*Help→Help Contents→ACE Guide*».

ΟΡΟΛΟΓΙΑ

Ξενόγλωσσος Όρος	Ελληνικός Όρος
abstract syntax	αφηρημένη σύνταξη
aggregation association	συσσωρευτική συσχέτιση
code folding	αναδίπλωση κώδικα
code generator	γεννήτορας κώδικα
compiler	μεταγλωττιστής
concrete syntax	διακριτή σύνταξη
context-free grammar	γραμματική χωρίς συμφραζόμενα
cost estimator	εκτιμητής κόστους
debugger	αποσφαλματωτής
directed labeled graph	κατευθυνόμενος γράφος με ετικέτες
documentation generator	γεννήτορας τεκμηρίωσης
domain expert	ειδικός πεδίου
domain framework	πλαίσιο εκτέλεσης
domain-specific language	γλώσσα ειδικού σκοπού
domain-specific language engineering	μηχανική γλωσσών ειδικού σκοπού
domain-specific visual language	γραφική γλώσσα ειδικού σκοπού
end user	τελικός χρήστης
external language	εξωτερική γλώσσα
general purpose language	γλώσσα γενικού σκοπού
graph grammar	γραφική γραμματική

graphical user interface	γραφική διασύνδεση χρήστη
Integrated Development Environment	ολοκληρωμένο περιβάλλον ανάπτυξης
internal language	εσωτερική γλώσσα
language cacophony problem	πρόβλημα κακοφωνία γλώσσών
language descriptions	περιγραφές γλωσσών
metaclass	μετακλάση
metamodel	μεταμοντέλο
model-driven software development	ανάπτυξη λογισμικού βασισμένου σε μοντέλα
model transformer	μετατροπέας μοντέλων
model verifier	επικυρωτής μοντέλων
modeling concept	έννοια μοντελοποίησης
modeling framework	πλαίσιο μοντελοποίησης
plugin	άρθρωμα
production rule	κανόνας παραγωγής
refactoring	αναμόρφωση κώδικα
repository	αποθετήριο
reverse engineering tool	εργαλείο αντίστροφης μηχανικής
semantics	σημασιολογία
software product line	γραμμή παραγωγής λογισμικού
target platform	πλατφόρμα στόχος
template language	γλώσσα προτύπων

textual editor	επεξεργαστής κειμένου
tool generator	γεννήτορας εργαλείων
tool vendor	κατασκευαστής εργαλείων
version control tool	εργαλείο διαχείρισης εκδόσεων
virtual machine	εικονική μηχανή
visitor pattern	πρότυπο επισκέπτη
visual editor	γραφικός επεξεργαστής

ΠΙΝΑΚΑΣ ΣΥΝΤΜΗΣΕΩΝ – ΑΡΚΤΙΚΟΛΕΞΩΝ

Αρκτικόλεξο	Λεκτική Περιγραφή
ANTLR	ANother Tool for Language Recognition
DDF	DSL Debugger Framework
DSM	Domain Specific Modeling
DSVL	Domain-Specific Visual Language
EBNF	Extended Backus-Naur Form
EMF	Eclipse Modeling Framework
EMP	Eclipse Modeling Project
GEF	Graphical Editing Framework
GME	Generic Modeling Environment
GMF	Graphical Modeling Framework
GPL	General Purpose Language
GUI	Graphical User Interface
IDE	Integrated Development Environment
JET	Java Emitter Templates
MDA	Model Driven Architecture
MDSD	Model-Driven Software Development
MWE	Model Workflow Engine
OCL	Object Constraint Language
OMG	Object Management Group
OSGi	Open Service Gateway Initiative

PDE	Plug-in Development Environment
TMF	Textual Modeling Framework
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XSD	XML Schema Definition

ΑΝΑΦΟΡΕΣ

1. S. Kelly and J.P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons, New Jersey, 2008.
2. R.C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, the eclipse series, Addison-Wesley, Pearson Education, 2009.
3. D. Steinberg, F. Budinsky, M. Paternostro and E. Merks, EMF: Eclipse Modeling Framework, 2nd edition, the eclipse series, Addison-Wesley, Pearson Education, 2009.
4. E. Clayberg and D. Rubel, Eclipse Plug-ins, 3rd edition, the eclipse series, Addison-Wesley, Pearson Education, 2009.
5. A. Kleppe, Software Language Engineering: Creating Domain-Specific Languages Using Metamodels, Addison-Wesley, Pearson Education, 2009.
6. T. Stahl, M. Volter, J. Bettin, A. Haase and S. Helsen, Model-Driven Software Development. John Wiley & Sons, 2005.
7. M. Scheidgen, Describing Computer Languages: Meta-languages to describe languages, and meta-tools to automatically create language tools, Ph.D Thesis, Humboldt-Univeritat zu Berlin, August 2008.
8. A.V. Aho, M.S. Lam, R. Sethi and J.D. Ullman, Compilers: Principles, Techniques & Tools, 2nd edition, Addison-Wesley, Pearson Education, 2007.
9. M. Wimmer and G. Kramler, Bridging Grammarware and Modelware, MoDELS Satellite Events. Lecture Notes in Computer Science, vol. 3844, pp. 159-168, Springer-Verlag, 2006.
10. ISO. ISO/IEC 14977:1996(E), Information technology – Syntactic metalanguage – Extended BNF, 1996.
11. N. Wirth, What can we do about the unnecessary diversity of notation for syntactic definitions, Communications of the ACM, 20(11), November 1997.
12. M. Alamen and I. Porres, A Relation Between Context-Free Grammars and Meta Object Facility Metamodels. Technical report, Turku Centre for Computer Science, 2003.

13. R. Seeley. (2004). ADT at Gartner ITxpo: Gates sees more modeling, less coding. Application Development Trends 03/30/2004. Retrieved September 04, 2009 from http://adtmag.com/articles/2004/03/30/adt-at-gartner-itxpo-gates-sees-more-modeling-less-coding.aspx .

14. M. Mernik, J. Heering and A.M. Sloane, When and How To Develop Domain-Specific Languages. ACM Computing Surveys, vol. 37, no. 4, pp. 316-344, December 2005.

15. M. Scheidgen, Textual Modelling Embedded into Graphical Modelling. Lecture Notes in Computer Science, vol. 5095, pp.152-168, Springer-Verlag, 2008.

16. L. Kats, K. Kalleberg and E. Visser, Domain-Specific Languages for Composable Editor plugins. Ninth Workshop on Language Descriptions, Tools, and Applications (LDTA '09), March 2009.

17. S. Kelly and R. Pohjonen, Worst Practices for Domain-Specific Modeling. IEEE Software, vol. 26, no. 4, July/August 2009.

18. J. Bezivin, On the unification power of models. Software and Systems Modeling, no. 4, pp. 171-188, (2005).

19. P.J. Landin, The Next 700 Programming Languages. Communications of the ACM, vol 9, no. 3, 1966, pp. 157.166.

20. J. Warmer, A Model Driven Software Factory Using Domain Specific Languages. Lecture Notes in Computer Science, vol. 4530, pp.194-203, 2007.

21. Fowler M., Beck K., Brant J., Opdyke W.F. and Roberts D.B., Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional (1999).

22. Mens T. and Tourwe, T., A survey of software refactoring. IEEE Trans. Software Engineering. no. 30 (2004).

23. Lammel R., Grammar adaptation. Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, Lecture Notes in Computer Science, vol. 2021, pp. 550–570, Springer-Verlag, 2001.

24. Wachsmuth G., Metamodel Adaptation and Model Co-adaptation. 21st European Conference on Object-Oriented Programming (ECOOP'07). Lecture Notes in Computer Science, vol. 4609, Springer-Verlag, 2007.

25. Kleppe A., Towards the Generation of a Text-Based IDE from a Language Metamodel. ECMDA-FA, pp. 114-129, 2007.

26. Bardohl R., GenGEd: Visual Definition of Visual Languages based on Algebraic graph Transformation. PhD thesis, TU Berlin, Berlin, Germany (1999).

27. Minas M., Generating meta-model-based freehand editors. In: Proceedings of the third International workshop on graph based tools, 2006, EASST, pp. 1–11, September 2006.

28. H. Ehrig, “Introduction to the Algebraic Theory of Graph Grammars,” Proc. Int’l Workshop Graph-Grammars and Their Application to Computer Science and Biology, V. Claus, H. Ehrig, and G. Rozenberg, eds., LNCS 73, Springer-Verlag, 1979.

29. Alanen M. and Porres I., A Relation Between Context-Free Grammars and Meta Object Facility Metamodels. Technical Report 606, TUCS, 2004.

30. H. Wu, J. Gray and M. Mernik, Debugging Domain-Specific Languages in Eclipse, *The 20th Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, San Diego, CA, October 2005.

31. M. Weiser: “The computer for the 21st century”, Scientific American, pp. 66-75, September 1999.

32. J.D Rivieres and J. Wiegand, Eclipse: A platform for integrating development tools, IBM Systems Journal, 43(2).

33. V. Nomikos and K. Kolomvatsos, Documentation of the IPAC Application Description Language, Technical Report, Department of Informatics and Telecommunications, University of Athens, September 2009.

34. M. Fowler, Language Workbenches: The Killer-App for Domain Specific Languages? Retrieved September 04, 2009 from http://martinfowler.com/articles/languageWorkbench.html .

35. ANTLR – ANother Tool for Language Recognition Website (n.d). Retrieved September 15, 2009 from http://www.antlr.org/ .

36. Java Compiler Compiler (JavaCC), Home page. Retrieved September 15, 2009 from https://javacc.dev.java.net/ .

37. SableCC, Home page. Retrieved September 15, 2009 from http://sablecc.org/ .

38. S.C. Johnson, Yacc – Yet another compiler compiler. Technical report CSTR 32, Bell Telephone Labs, July, 1974.

39. Object Constraint Language (OCL) specification 2.0. Formal/06-05-01.

40. M. Scarpino, S. Holder, S. Ng, and Laurent Mihalkovic, SWT/JFace in Action. Manning Publications, 2004.

41. SWT: The Standard Widget Toolkit, Home Page. Retrieved October 5, 2009 from http://www.eclipse.org/swt/ .

42. OSGi – The Dynamic System Module for Java. Retrieved September 29, 2009 from http://www.osgi.org/About/Technology .

43. OSGi in Practice. Retrieved September 30, 2009 from http://neilbartlett.name/blog/osgibook/ .

44. Object Management Group (OMG), Home page. Retrieved October 2, 2009 from http://www.omg.org/ .

45. OMG. Meta Object Facility (MOF) 2.0 Core Specification. Retrieved September 15, 2009 from http://www.omg.org/docs/formal/06-01-01.pdf .

46. OMG. XML Metadata Interchange, version 1.2, January 2002. Retrieved September 21, 2009 from http://www.omg.org/.

47. The Eclipse Platform, Home page. Retrieved September 22, 2009 from http://www.eclipse.org/ .

48. NetBeans, Home Page. Retrieved September 22, 2009 from http://netbeans.org/ .

49. Eclipse Modeling Project (EMP), Home page. Retrieved September 22, 2009 from http://www.eclipse.org/modeling/.

50. Eclipse Modeling Framework (EMF), Home page. Retrieved September 22, 2009 from http://www.eclipse.org/modeling/emf/ .

51. Graphical Editing Framework (GEF), Home page. Retrieved September 22, 2009 from http://www.eclipse.org/gef/ .

52. Graphical Modeling Framework (GMF), Home page. Retrieved September 22, 2009 from http://www.eclipse.org/modeling/gmf/ .

53. Model To Text (M2T), Home page. Retrieved September 22, 2009 from http://www.eclipse.org/modeling/m2t/?project=xpand

54. Modeling Workflow Engine (MWE), Home page. Retrieved September 22, 2009 from http://wiki.eclipse.org/Modeling_Workflow_Engine_(MWE) .

55. Textual Modeling Framework (TMF), Home page. Retrieved September 22, 2009 from http://www.eclipse.org/modeling/tmf/ .

56. Xtext, Home page. Retrieved September 22, 2009 from http://www.eclipse.org/Xtext/ .

57. openArchitectureWare (oAW), Home page. Retrieved September 24, 2009 from http://www.openarchitectureware.org/ .

58. Plug-in Development Environment (PDE), Home page. Retrieved September 12, 2009 from http://www.eclipse.org/pde/ .

59. DSM Forum, Home page. Retrieved September 27, 2009 from http://www.dsmforum.org/ .

60. MetaCase, Home page. Retrieved September 26, 2009 from http://www.metacase.com/ .

61. Getting Started with DSL Tools. Retrieved September 26, 2009 from http://msdn.microsoft.com/en-us/vsx/cc677260.aspx .

62. Meta-Object Facility (MOF). Retrieved September 30, 2009 from http://en.wikipedia.org/wiki/Meta-Object_Facility.

63. The Generic Modeling Environment (GME), Home page. Retrieved September 18, 2009 from http://w3.isis.vanderbilt.edu/projects/gme/index.html.

64. The Apache Ant Project, Home page. Retrieved September 29, 2009 from http://ant.apache.org/ .