Smv2Cdl

Veritech Project

אייל סונסינו
"הטכניון" – מכון טכנולוגי לישראל
קיץ 2001

תוכן הענינים:

הקדמה
SMV – תרשימי מחלקות:
תרשים מחלקות כללי
תרשים מחלקות Expression
תרשים מחלקות Type
תרשים מחלקות Declaration
תרשים מחלקות List
מחלקות ה-SMV
Atom
Assignment
template<class T> List
template<class T> Iterator
זיכרון
Clone
מחלקות ה-Compiler
Binder
Collector
TypesReplace
DefineInline
CDLBuilder
TransitionBuilder

Unsupported features

הקדמה

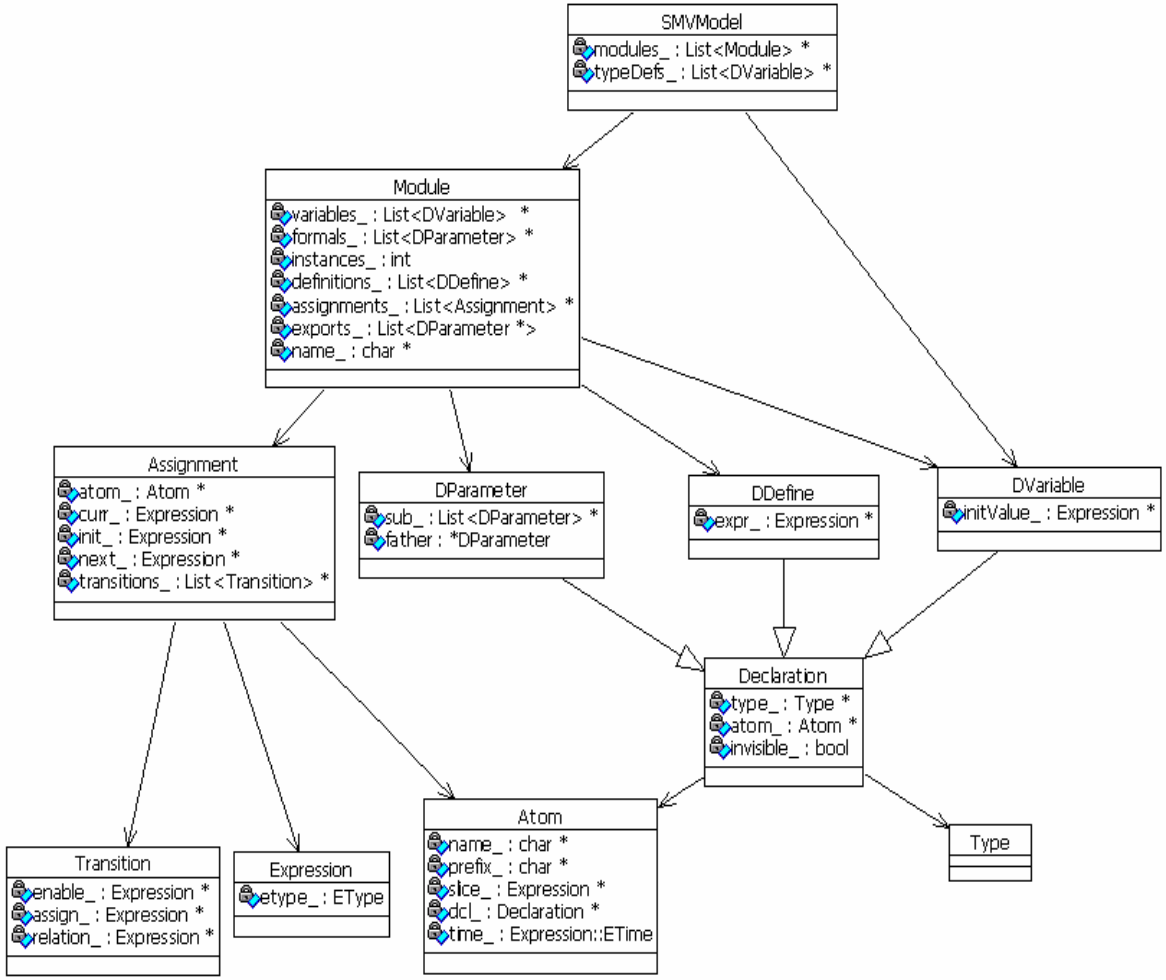
smv2cdl הוא הרכיב בפרוייקט veritech המבצע תרגומים מ – SMV אל שפת הגרעין – CDL.
רכיב זה מורכב משתי חבילות:

1. SMV – ספריה המכילה מחלקות עבור המבנים השונים ב-SMV כגון ביטויים, הצהרות מודולים וכד'. ספריה זו אינה מכילה פונקצית main ולכן אינה יוצרת קובץ הרצה.
בעזרת ספריה זו בונים מבנה נתונים בזיכרון המייצג את המודול המתואר בקובץ הקלט הכתוב בפורמט SMV.
בין יתר המחלקות בספרייה המייצגות מבנים אופיינים בשפת SMV קיימות גם המחלקות List ו-Iterator (שתיהן מחלקות template), SMVVisitor המשמש interface עבור כל מחלקה הממשת visitor (*).
ובנוסף המחלקה SMVPublisher שביכולתה להדפיס מודל SMV הנמצא בזיכרון לקובץ כלשהו בפורמט SMV סטנדרטי.

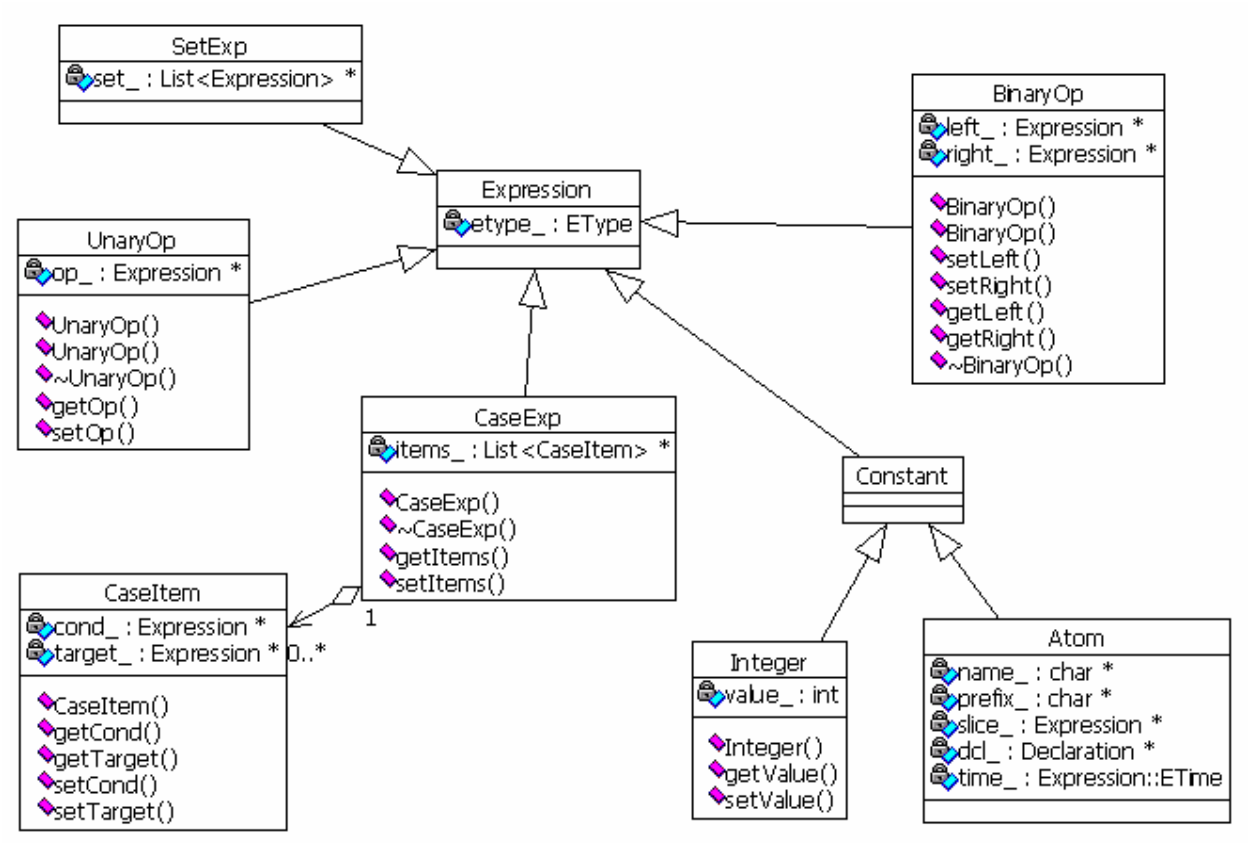
2. Compiler – בחבילה זו נמצא הקובץ Main.cpp (שבו נמצאת הפונקציה main).
תפקיד חבילה זו הוא לקרוא את מודל ה-SMV מהקובץ (parsing) ובניית מודל ה-SMV המתאים בזיכרון (בעזרת שימוש במחלקות ה-SMV כמובן).
לאחר סיום הפענוח והבנייה, קיים בידינו מודל SMV בזיכרון המקביל למודל המתואר בקובץ.
בשלב הבא מתבצעות על המודל טרנספורמציות למודל שיוכל לתאר מודל CDL שקול.
המחלקות השונות בחבילת ה-Compiler אחראיות על ביצוע טרנספורמציות אלו.
חלק מהטרנספורמציות הוא למשל מיקום מחדש של כל דקלרציות (Collector), הוצאת טיפוסים מורכבים החוצה (TypesReplace), מיפוי מחדש של אינדקסים בגישה למערכים, יצירה מבנית (CDLBuilder), שיטוח ביטויי Case ויצירת Transitions (TransitionBuilder) ועוד'.
מחלקה נוספת היא המחלקה CDLPublisher שתפקידה להדפיס את המודל הנמצא בזיכרון בפורמט הסטנדרטי של CDL.

* במחלקות רבות בקומפיילר נעשה שימוש ב-visitor pattern על כן, מומלץ מאוד לכל המעונין להבין את אופן המימוש להכיר את ה-pattern.
ניתן למצוא הסבר מלא על ה-visitor בספר "Design patterns"

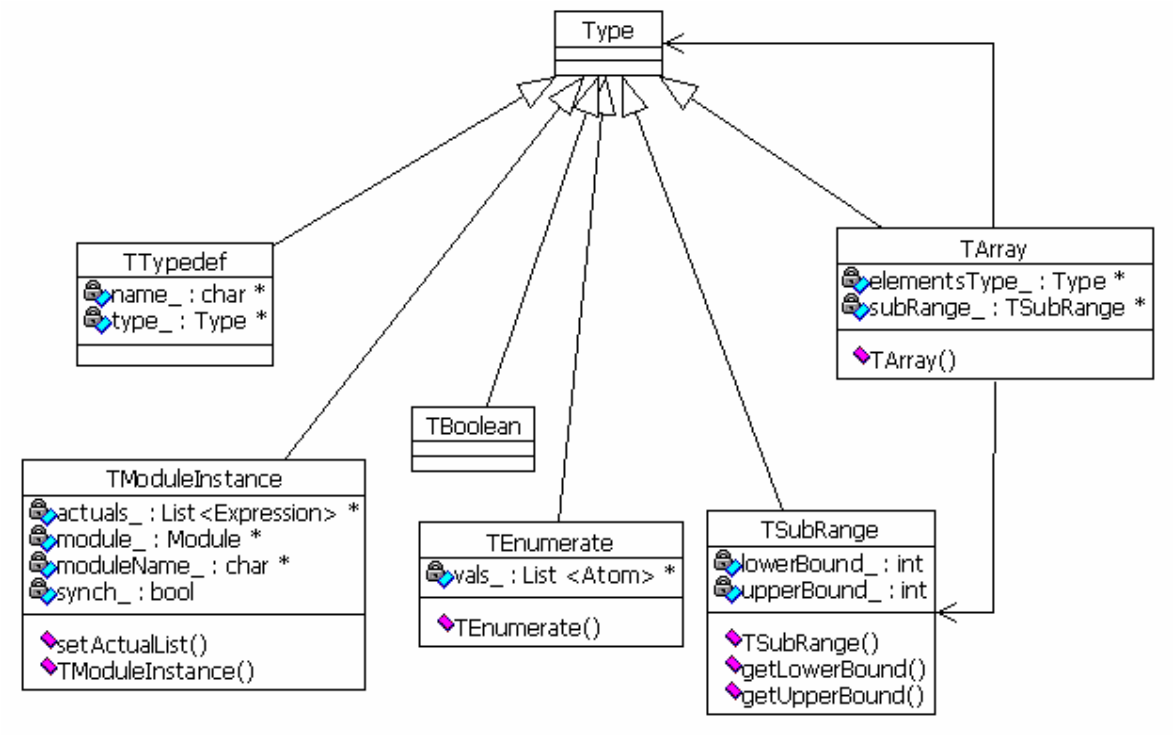
SMV – General Class Diagram



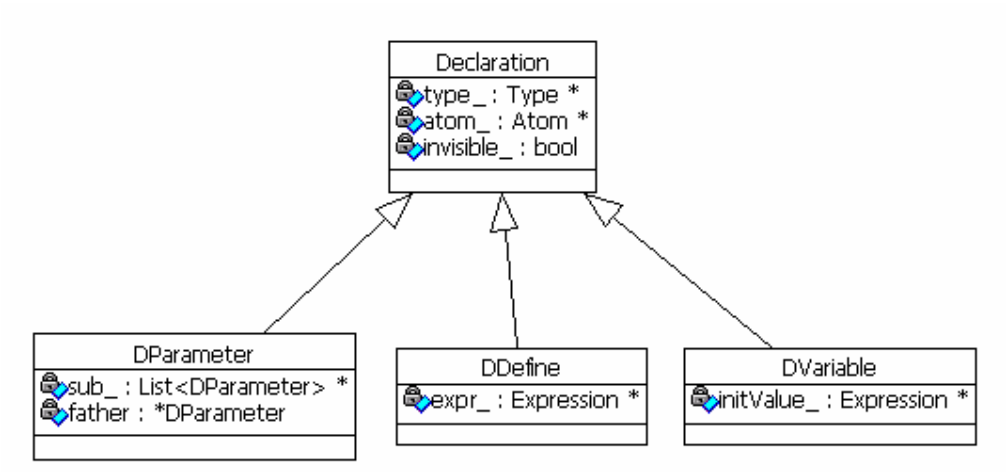
SMV – Expression Class Diagram



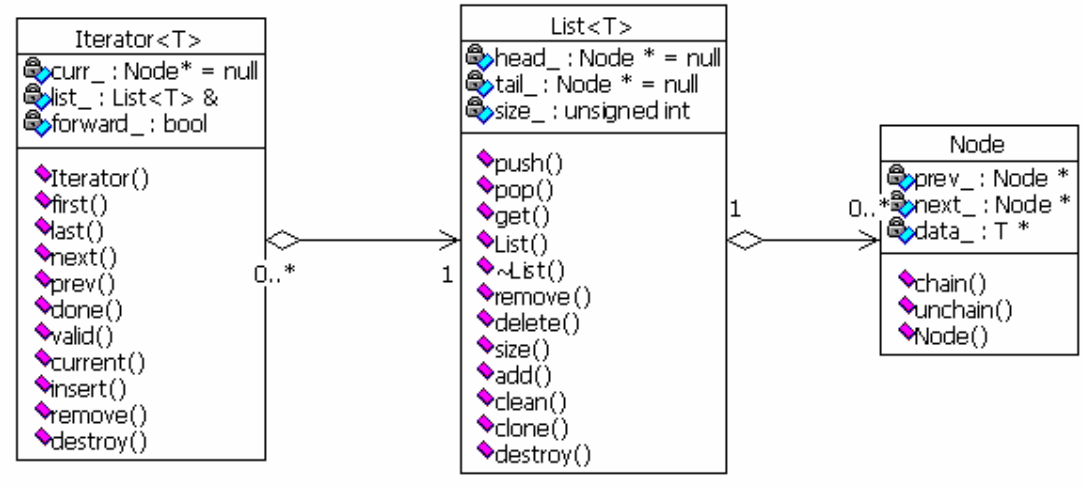
SMV – Type Class Diagram



SMV – Declaration Class Diagram



SMV – List Class Diagram



מחלקות ה-SMV

כפי שכבר צויין, חבילת ה-SMV מיועדת לספק מבנה נתונים שייצג מודל SMV.
המחלקה SMVModel מחזיקה את כל המודל, ניתן לראות שהיא מחזיקה רשימה של מודולים (_modules) וכן, רשימה של הגדרות טיפוסים (_typedefs).
כל מודול מיוצג ע"י המחלקה Module. עבור כל מודול נשמרת רשימת הפרמטרים הפורמליים, ההצהרות על משתנים והצהרות Define, רשימת ההשמות וכו'.

סוגי הצהרות - Declaration:

1. DParameter – המשמש לייצוג פרמטר פורמלי.
2. DDefine – משמש לייצוג הצהרה על DEFINE.
3. DVariable – משמש לייצוג הצהרה על משתנה מטיפוס כלשהו.

סוגי ביטויים - Expression:

1. CaseExp – מייצג ביטוי Case ומחזיק רשימה של CaseItem.
2. SetExp – מייצג ביטוי קבוצה, למשל: {1,56,3}
3. BinaryOp – מייצג ביטוי בעל שני אופרנדים למשל: + , * .
4. UnaryOp – מייצג ביטוי בעל אופרנד אחד למשל () או !.
5. Constant – קבוע כלשהו (המחלקה מהווה רק Interface עבור Atom ו-Integer)
   א. Atom – מייצג **או** מזהה **או** ערך של רשימת Enumerated – מחזיק שם.
   ב. Integer – מייצג קבוע מספרי או קבוע בוליאני – מחזיק ערך.

מכיוון שייתכנו סוגים שונים של ביטויים עבור אותה מחלקה, סוג הביטוי המדויק נשמר במשתנה חבר _etype. למשל:
עבור BinaryOp מסוג +, etype_=Expression::E_PLUS
עבור BinaryOp מסוג -, etype_=Expression::E_MINUS
עבור UnaryOp מסוג !, etype_=Expression::E_NOT
אך עבור CaseExp קיימת האפשרות יחידה: etype_=Expression::E_CASE

סוגי טיפוסים – Type:

1. Tboolean –מייצג טיפוס בוליאני.
2. TSubRange – מייצג טיפוס תת-תחום כגון 1..5.
3. TEnumerate – מייצג רשימת Enumerated.
4. TArray – מייצג מערך, לשם כך מחזיק את התחום של האינדקס ואת הטיפוס של האיברים.
5. TTypedef – מיצג טיפוס שהוצהר ע"י Typedef
6. TModuleInstance – מייצג מופע של מודול, לשם כך מחזיק את רשימת הפרמטרים האקטואלים וכן את שם המודול ואת סוג הסינכרון (סינכרוני עבור מודול רגיל ואסינכרוני עבור תהליך).

כל המחלקות ממשות מתודה – accept שייעודה הוא השימוש ב-visitor.

במחלקות List, Iterator, Node נעשה שימוש רב הן במודל ה-SMV והן במהלך יצירת קוד ה-CDL.
הן יפורטו בהרחבה בהמשך.

! רוב המבנים הם מבנים פשוטים ביותר. אטום והשמה הם מרוכבים מעט יותר ועל כן ארחיב עבורם.

<u>אטום (Atom):</u>
כפי שצוין קודם, אטום יכול לשמש גם בתור מזהה וגם בתור ערך של רשימת Enumerated.
במידה והאטום מתפקד כערך של רשימת Enumarated, יוצב: etype_=Expression::E_ENUM
אחרת – האטום מייצג מזהה, עבורו etype_=Expression::E_ATOM
כאשר אטום מייצג מזהה, קיימות תכונות נוספות המתאימות עבורו:

- _time: מציין לאיזה זמן מתייחס האטום.למשל:
  עבור init(a) נקבל time_=Expression::T_INIT
  עבור next(a) נקבל time_=Expression::T_NEXT
  ועבור a נקבל time_=Expression::T_CURR
- _slice: ביטוי המציין גישה לאיבר במערך.
  למשל עבור x[5+d], _slice יצביע לביטוי 5+d.
  במידה ואין גישה לאיבר במערך, _slice יקבל את הערך NULL.
- _dcl: מצביע להצהרה של המזהה. כברירת מחדל – NULL.

<u>השמה (Assignment):</u>
השמה היא מבנה המכיל את הערכים שהוגדרו עבור init ו-next (או ערך תמידי ב-curr) של אטום מסוים.
כלומר אם בקובץ ה-SMV הוגדר:

```
init(a):=3+b;
next(a):=d-1;
```

אזי תיוצר השמה עבור האטום a כאשר ביטוי ה-init יהיה 3+b וביטוי ה-next יהיה d-1.
במידה והוגדר:

```
a:=g*3;
```

אזי ההשמה שתיוצר עבור האטום a תכיל ביטוי curr – g*3.
בכל מקרה מצב שבו קיים ערך גם עבור init או next וגם עבור curr אינו חוקי.

לכל השמה קיים מצביע לרשימת Transition (transitions_) המאותחלת ל-NULL. בעת יצירת מודל CDL מיוצרת לכל השמה רשימת Transition מתאימה המושמת באותו מצביע. כך בסיום העיבוד ניתן לעבור על כל ההשמות ולהדפיס את רשימת ה-Transition בפורמט CDL.

<u>template <class T> class List</u>

מחלקת ה-List<T> מממשת רשימה מקשרת דו כיוונית המחזיקה מצביעים לאובייקטים מטיפוס T.
למחלקה קיימת תת-מחלקה פרטית - template<class T> List::Node. מחלקה זו מממשת איבר ברשימה.
כל איבר מחזיק מצביע לאיבר הקודם (_prev), מצביע לאיבר הבא (_next) ומצביע למידע (_data).
בכל רשימה קיים איבר לכל נתון המוחזק בה ובנוסף גם איבר עבור ראש הרשימה ואיבר עבור זנבה. תוספת זו מקלה את הטיפול במקרי קצה.
מצביע ה-prev של האיבר המייצג את ראש הרשימה מאותחל לראש הרשימה בעצמו. באופן דומה עבור מצביע ה-next של הזנב. שדה ה-data עבור הראש והזנב הוא NULL.

ב-List ממומשות כל המתודות הרגילות הצפויות להופיע ב-List כדון push, pop, add וכדו' הסברים מלאים על כל מתודה ניתן למצוא בקבצי ה-header.

יש לשים לב כי ייתכנו שני סוגי רשימות:

1. רשימה המחזיקה בבעלות על האובייקטים עליהם היא מצביעה (owned).

2. רשימה שאינה בעלת האובייקטים עליהם היא מצביעה (כלומר קיים אובייקט אחר המשמש בעלים של האובייקטים עליהם הרשימה מצביעה ושהוא אחראי למחיקתם).

עבור המקרה הראשון, לפני שמוחקים את הרשימה יש צורך למחוק גם את האובייקטים המוצבעים על ידה.
לשם כך קיימת המתודה List<T>::destroy() שתפקידה למחוק את כל האיברים ברשימה (מלבד הראש והזנב) ואת האובייקטים המוצבעים (_data) על ידי כל אחד מהם.
אין כל אכיפה של מחיקה ולכן אחריות המחיקה (קריאה ל-destroy() לפני delete) היא של המתכנת המשתמש ברשימה. אי מחיקה של אובייקטים תסתיים בדליפת זיכרון.
במקביל ל- destroy() קיימת המתודה List<T>::clean() שתפקידה למחוק את כל האיברים ברשימה (שוב מלבד הראש והזנב) אך הפעם ללא מחיקת האובייקטים המוצבעים.
בשני המקרים ניתן לבצע שימוש חוזר ברשימה לאחר מכן.

במידה וברצון המתכנת להשתמש באחת המתודות הבאות:

```
T* List<T>::get(const char *str)
T* List<T>::remove(const char *str)
bool List<T>::destroy(const char *str)
```

עליו לספק את המתודה:

```
bool T::operator = =(const char *str)
```

מתודה זו מופעלת על מנת לאתר את האובייקט המתאים (הראשון) ברשימה.

ועבור השימוש במתודה:

```
List<T> *List<T>::clone()
```

עליו לספק את המתודה:

```
T* T::clone()
```

(ראה פרק clone)

template <class T> class Iterator

מחלקת ה-Iterator משמשת לביצוע מעבר סדרתי על List<T>. איטרציה כזו יכולה להתבצע קדימה (מהראש לכיוון הזנב) או אחורה (מהזנב לכיוון הראש). כיוון ההתקדמות נקבע ע"י פרמטר ב-C`tor (אשר מאותחל כברירת מחדל לסריקה קדימה).
הפרמטר הראשון ב-C`tor הוא List<T> & ולכן יש לשים לב להעביר רשימה חוקית ולא (NULL)*.
ההתקדמות בסריקה תתבצע בעזרת המתודות prev()/next() בהתאם לכיוון המבוקש.

כאשר מחזיקיIterator עבור רשימה מסוימת, אסור לבצע פעולות עדכון ע"י המתודות של List. פעולות כאלו עלולות לפגוע בעקביות המידע.
← לשם ביצוע פעולות עדכון רשימה במהלך איטרציה, קיימות מתודות מתאימות במחלקה Iterator כגון: insert(),remove(), destroy()המתייחסות לאיבר הנוכחי המוצבע ע"י ה-Iterator. הסברים מלאים ניתן כמובן למצוא בקבצי ה-header.

זיכרון

כפי שכבר הוזכר בפרק העוסק ב-List, לכל אובייקט שנוצר בזיכרון קיים בעלים. המטרה היא מניעת דליפת זיכרון. כאשר ישוחרר הזיכרון של אובייקט, ישוחרר גם הזיכרון של כל האובייקטים הנמצאים בבעלותו.
לדוגמא Atom הוא הבעלים של החברים name_, prefix_, slice_ אך אינו הבעלים של dcl_ - ההצהרה על האטום (הבעלים שלה הוא המודול בו היא נמצאת). לכן כאשר נבצע מחיקה של ה-Atom יש צורך למחוק את name_, prefix_, slice_ אך לא נמחק את dcl_.

קיימים מצבים בהם יש להיזהר מרמיסה של ערכים שהיו בבעלות אובייקט.
לדוגמא:
עבור המחלקה BinaryOp (המממשת ביטוי בעל שני אופרנדים) קיימים שני החברים: left_ ו-right_ שניהם מטיפוס מצביע לביטוי ושניהם בבעלותה.
לפיכך בעת מחיקת BinaryOp תתבצע מחיקה גם ל-left_ ול-right_ .
כעת נתבונן במצב בו קיים ביטוי BinaryOp - e, ולו שני ארגומנטים (שאינם NULL).
במצב כזה, אם נרצה לבצע:

e→setLeft(ביטוי חדש כלשהו);

נרמוס את הערך שהיה ב-left_ ואף אחד אחר לא ידאג לשחרורו.
לפיכך, לפני השמת ערכים חדשים עבור חברים שהם בבעלות האובייקט יש למחוק אותם תחילה. (דוגמאות לכך ניתן למצוא במחלקה-DefineReplace)
במקרים מסוימים נרצה לעשות שימוש באותם אובייקטים מוצבעים (מיחזור אובייקטים). במקרים כאלה נשמור את הערכים לפני ההשמה ואז נבצע את ההשמה.

Clone

לעיתים רבות קיים הצורך לבצע deep copy עבור אובייקטים, כלומר העתקת האובייקט וכל העץ המוצבע על ידו.
לשם כך קיימת המתודה clone הממומשת בכל מחלקה ותפקידה להחזיר אובייקט חדש אשר הינו העתק של האובייקט הנוכחי ושמחזיק clone של האובייקטים המוצבעים על ידיו.

כמובן שבמקרים בהם גרף ההצבעות מעגלי תתעורר בעיה – ההעתקה לא תסתיים לעולם.
לכן clone עבור מחלקה כלשהי יקרא ל-clone רק עבור החברים אשר בבעלות המחלקה. לכל היתר יתבצע shallow copy כלומר השמה של אותם הערכים.

לדוגמא Atom הוא הבעלים של החברים name_, prefix_, slice_ אך אינו הבעלים של dcl_.
לכן ב- Atom *Atom::clone() נקרא ל-clone()→slice אך לא תתבצע קריאה ל-clone()→dcl_.

מחלקות ה-Compiler

להלן המחלקות בהן נעשה שימוש בספרית ה-Compiler, אופן פעולתן יפורט בהמשך.

- ErrorManager – ריכוז הטיפול בהודעות שגיאה (errors) ואזהרות (warnings).
- CmdLine – פענוח הארגומנטים בשורת הפקודה.
- Binder – קשירת משתנים והצהרות.
- TypesReplace – החלפת טיפוסים.
- Collector – חישוב רשימת הפרמטרים הפורמליים והאקטואליים המפורשת עבור כל מודול.
- DefineInline – החלפת מזהים המוגדרים כ-DEFINE.
- CDLBuilder – בניית מודל CDL ממודל SMV
- TransitionBuilder – בניית רשימת Transition עבור השמה
- CDLPublisher – הדפסת מודל CDL הנמצא בזיכרון לקובץ כלשהו בפורמט CDL סטנדרטי.

ErrorManager

מחלקה כללית שנותנת שירותים לכל הרכיבים בקומפיילר.
קיים אובייקט גלובלי אחד מטיפוס זה (מוצהר בקובץ Main.cpp) ומוצהר כ- extern בקובץ ErrorManager.h
מחלקה זו מרכזת את הטיפול בהודעות שגיאה ואזהרה.
הדפסת ההודעות מתבצעת לקובץ רישום (Logfile) וכן לערוץ השגיאות (stderr).
עם סיום הריצה מציגה סיכום השגיאות והאזהרות.

CmdLine

מחלקה המשמשת את הפונקציה הראשית main. באחריותה פענוח הפרמטרים בשורת ההפעלה.

Binder

במחלקה זו נעשה שימוש בשלב מוקדם מאוד של הניתוח של המודל.
תפקידה הוא לקשור כל מזהה עם ההצהרה המתאימה – הצהרת משתנה , הצהרת מאקרו, פרמטר פורמלי.
כמו כן כל מופע של מודול נקשר עם ההגדרה שלו.
כפי שכבר הוסבר קודם – אטום משמש לייצוג מזהה ו/או ערך ברשימת Enumerated בשלב ה-Binding מנסים למצוא לכל אטום בביטוי הצהרה מתאימה. במידה והצהרה כזו לא נמצאה, מניחים שמדובר בערך Enumerated.
אטומים המופיעים בצד שמאל של השמה יכולים להיות אך ורק מזהים ולכן במידה ולא נמצאה הצהרה מתאימה עבור אטום המופיע בצד שמאל תוצג הודעת שגיאה.
במהלך קשירת המודולים מתבצעת בדיקה שלכל מופע של מודול אכן קיים מודול מתאים ושמספר הארגומנטים שהועבר אליו שווה למספר הארגומנטים בהגדרת המודול.
בסיום התהליך לכל מודול מתעדכן מספר המופעים שלו (בערך זה נעשה שימוש בעת המיון טופולוגי)

TypesReplace

מחלקה זו מטפלת בהחלפת/התאמת הטיפוסים – מטיפוסי SMV לטיפוסי CDL.
השוני העיקרי שקיים בין SMV ל-CDL מבחינת טיפוסים הוא שהטיפוסים תת תחום (TsubRange) ורשימת ערכים (Enumerated) יכולים להופיע ב-CDL רק בראשית הקובץ כשהם מוגדרים בעזרת Typedef. השימוש בהם נעשה על ידי השימוש בשם החדש (כפי שהוגדר בהצהרת ה-Typedef).
לשם הסבת המודל למודל CDL מקביל, מתבצע מעבר על כל ההצהרות על משתנים במודל והחלפת הטיפוסים עבור משתנים שהטיפוס שלהם בעייתי.
עבור משתנה בשם: id_name שהוגדר במודול: module_name בעל טיפוס תת תחום או רשימת ערכים יתווסף טיפוס Typedef חדש ששמו: T#module_name#id_name והטיפוס אליו הוא מקושר יהיה הטיפוס המקורי שהוגדר עבור המשתנה.
הטיפוס החדש של המשתנה יוגדר כ -T#module_name#id_name .

DefineInline

מחלקה זו מספקת שירות שגורם להחלפת כל המופעים של מאקרו (DEFINE) בביטויים המכילים קבועים ומשתנים בלבד.
ההחלפה מתבצעת באופן רקורסיבי כך שאם מאקרו משתמש בהגדרתו במאקרו אחר, תתבצע החלפה גם עבורו.
כמו כן, מתבצעת בדיקה של הגדרות מעגליות. במידה וקיימות הגרות כאלו תוצג הודעת שגיאה מתאימה.

Collector

מחלקה זו באה לטפל בשלוש בעיות:

1. ב-SMV ניתן לצפות (observe) במשתנים שהוגדרו במודול אחד מתוך מודול אחר.
2. ב-SMV ניתן להעביר מודול שלם כפרמטר למודול אחר.
3. ב-SMV לא מציינים באופן מפורש את הטיפוסים של הפרמטרים הפורמליים של מודול.

בכדי לייצר קוד מתאים ב-CDL, יש צורך לתקן את רשימת הפרמטרים (אקטואליים ופורמליים) כך שהמשתנים בהם נעשה שימוש יועברו באופן מפורש. כמו כן, יש לקבוע טיפוסים עבור הפרמטרים הפורמליים.

מחלקה זו אחראית לחישוב רשימת היצוא (export list) של כל מודול וכן לרשימת sub-parameter של כל פרמטר פורמלי (האיבר החבר sub במחלקה DParameter).
המחלקה אוספת מידע חיוני זה כך שבאמצעותו, ניתן יהיה לקבוע את רשימת הפרמטרים הפורמליים והאקטואליים המפורשת עבור כל מודול. כלומר הממשק של כל מודול.

דוגמא 1:

```
Module X
        p : P();
        a : BOOLEAN;
ASSIGN
a:=p.d;


MODULE P
        d : BOOLEAN;
ASSIGN
        d:=TRUE;
```

במודול X מתבצעת צפייה בערכו של המשתנה הפנימי d שהוגדר ב-P, ולא הועבר בצורה מפורשת אל X.
הגישה אליו התאפשרה רק בגלל ש-p אשר הוגדר כמופע של P מוכר במודול.
עם סיום המעבר במקרה זה, ברשימת היצוא של המודול P יופיע d.

דוגמא 2:

```
Module R
        q : Q();
        s : S(q);
ASSIGN
: : :

MODULE Q
        x : BOOLEAN;
ASSIGN
        x:=TRUE;

MODULE S(e)
        t: BOOLEAN;
ASSIGN
        t:=e.x;
```

ראשית מתבצעת העברה של מודול כפרמטר למודול אחר – במקרה שלנו: q מועבר למופע של-S.
במודול S מתבצעת צפייה בערכו של המשתנה הפנימי x שהוגדר ב-e אשר הינו מופע של Q כלומר, במודול S מתבצעת צפייה במשתנה x שהוגדר במודול Q ולא הועבר בצורה מפורשת אל S.
הגישה אליו התאפשרה רק בגלל שמופע של Q הועבר אל S והתבצעה צפייה אל משתנה בתוך Q.
עם סיום המעבר במקרה זה, ברשימת היצוא של המודול Q יופיע x.
ברשימת תת-הפרמטרים של הפרמטר הפורמלי e במודול S יופיע x.

CDLBuilder

מחלקה זו מבצעת את השינויים האחרונים במעבר אל מודל CDL. זאת לאחר שכל יתר הטרנספורמציות על המודל כבר בוצעו (Binding,DefineInline,TypesReplace,Collector). עם סיום פעולתה, המודל יוכל לייצג מודל CDL.
תפקידה של המחלקה:

4. תיקון רשימת הפרמטרים הפורמליים בכל הגדרת מודול והאקטואליים במופעי המודול (בהתאם לרשימת הייצוא ותת הפרמטרים שחושבו בעזרת ה-Collector).
5. מיפוי מחדש של ביטויי מערך (Slice): מכיוון שב-SMV הגבול התחתון של מערך יכול להיות ערך כלשהו, וב-CDL הגבול התחתון מוגדר כ-0, יש לתקן את הגבולות עבור טיפוסי מערך. כמו כן, יש צורך למפות מחדש כל גישה לאיבר במערך כך שיתאים לגבולות החדשים.
6. עבור כל השמה תתבצע בניית מודול חדש ויצירת רשימת המעברים (Transition List) המתאימה עבורה (על ידי שימוש ב-TransitionBuilder).

TransitionBuilder

מחלקה זו בונה עבור השמה כלשהי את רשימת המעברים השקולה לה.
הנחת היסוד – הביטויים בהשמות נמצאים ב"צורה נורמלית".
ביטוי נמצא ב"צורה נורמלית" אם האב של כל צומת case בביטוי שאינו שורש הוא ביטוי case.

כלומר: !case, case+case אינם ביטויים בצורה נורמלית.
אך:

```
case
        cond1 : value1;
        cond2 : case
                  cond3 : value3;
                  cond4 : value4;
                esac;
        cond5 : value5;
        cond6 : value6;
esac;
```

הוא ביטוי בצורה נורמלית.