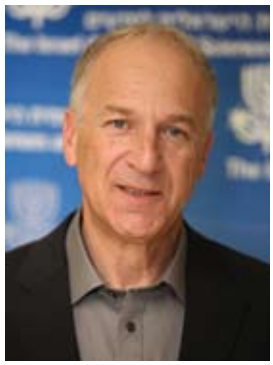
# תכנות בדרך הטבע

מאת **פרופ׳ דוד הראל**



מאמר זה מוקדש לזכרו של **פרופ׳ אמיר פנואלי** ז״ל, שנפטר בשנת תש״ע (2009). אמיר היה מדען דגול, חבר יקר, איש אשכולות, צנוע וחכם. הוא נבחר לאקדמיה הלאומית הישראלית למדעים בשנת תשס״א (2001).

## הקדמה

אם נשאל אנשי מקצוע בתחום מדעי המחשב ותכנות המחשבים ״מהו תכנות?״, יגידו רבים משהו כזה: תכנות הוא תהליך תכנוני ויישומי של הכנת ״מתכון״ (=אלגוריתם) בשביל מחשב, שכאשר הוא ״יורץ״ יבצע המחשב בדיוק את מה שהתכוון לו המתכנת או צוות התכנות. על פי הגדרה זו התכנות הוא פעילות שתוצאתה תכנית מחשב שאפשר להריץ אותה בו כדי לבצע מטלה מסוימת. זו הגדרה סבירה מאוד, אך הייתי רוצה להציע כאן הגדרה כללית יותר, שאינה קשורה בהכרח למחשב או לתכניות שרצות בו. לפי דעתי, תכנות במובן הכללי ביותר של המילה הוא התהליך שבעזרתו אנו גורמים לזולת לעשות את מה שאנו רוצים שיעשה.

במובן הכללי הזה אנו, בני האדם מן השורה, ״מתכנתים״ באופן טבעי לגמרי, יום יום, ומה שאנו ״מתכנתים״ הוא בני אדם אחרים, ולא בהכרח מחשב. אנו גורמים (או מנסים לגרום...) לילדינו, לתלמידינו, לעובדים שלנו אם אנו מנהלים, לפקידי הבנק שלנו, למלצרים במסעדה, לסוכני הנדל״ן שלנו וכו׳ לעשות דברים מסוימים. כלומר, אנו מנסים לגרום לאנשים אחרים להתנהג כפי שאנו רוצים שיתנהגו.

אנו עושים זאת מתוך שילוב של מגוון רחב של סוגי הנחיות: לפעמים נותנים הוראות מפורשות של ״עשה״ ושל ״אל תעשה״; לפעמים נותנים הנחיה כללית יותר ולפעמים רק מדגימים, בבחינת ״ממני תראו וכן תעשו״; לפעמים תוחמים התנהגות רצויה בגבולות מסוימים, ועוד. כמובן, לא כולנו מגדלים את ילדינו או מנהלים את עובדינו בדרך הטובה ביותר, ולא תמיד הדברים יוצאים כפי שהיינו רוצים. אבל מושג התכנות במובנו הכללי הוא בדיוק זה: לגרום לאחר להתנהג כפי שאנחנו רוצים. אפשר לראות הקבלה בין תכנות מחשב לתכנות הכללי יותר, האנושי, באיור 1.

1

המאמר הזה עוסק בכיוון מחקר שהחלום המנחה אותו הוא להגיע לתכנות פשוט, אינטואיטיבי וטבעי של מערכות ממוחשבות, שיהיה קרוב מאוד לדרך שבה אנו ״מתכנתים״ בני אדם, כלומר מקנים להם את ההתנהגויות הרצויות לנו.

## פיתוחן של מערכות מורכבות

נתרכז לפי שעה במערכות ממוחשבות ובתכנונן, ואחר כך נחזור לבני אנוש. באיור 2 אנו רואים תהליך פיתוח רצוי של מערכת מורכבת. כפי שאפשר לראות, הכול הולך חלק:

2

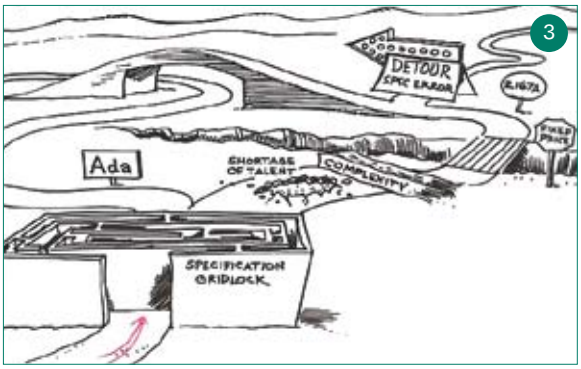
המערכת מפותחת בזמן, ופיתוחה עומד בתקציב; יש לה כל התכונות הדרושות למערכת מוצלחת, וכמובן החשוב ביותר הוא שהלקוח מקבל את המערכת ומרוצה ממנה. ואולם אם מסתכלים היטב באיור הזה רואים שאין בו אנשים כלל. הסיבה פשוטה מאוד: כולם נמצאים הרחק מעבר לאופק. פיתוח המערכת עלה יפה כל כך עד שכבר אין רואים את האנשים שבתהליך, והכול מרוצים.

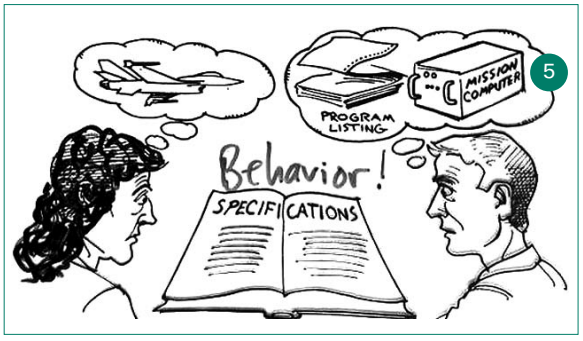
הבעיה היא שלא כך נראה עולם הפיתוח של מערכות מורכבות. לו היה פיתוח המערכות קל כל כך, לרוב אנשי המחשבים לא הייתה עבודה ומן הסתם הם היו נאלצים לעסוק בדברים אחרים. איור 3 מראה את המצב המצוי. כך נראה תהליך פיתוח אמתי, שיש בו בעיות, נפילות, עיכובים, אפשרות להיכנס למעגלים בלי יכולת לצאת מהם וכו'. מעניין שגם כאן אם נסתכל היטב נראה שאין אנשים. אבל הסיבה אחרת לגמרי: כולם תקועים באזור שנמצא בפינה השמאלית התחתונה של האיור, שאפשר לקרוא לו בעברית "מלכודת האפיון" או "מלכודת התכנון" – הם עדיין עסוקים בשאלות הקשורות להגדרה מה רוצים שהמערכת תעשה. באיור 4 רואים את האזור הזה במבט מקרוב. זהו מבוך שבו מסתובבים אנשים מהצוותים השונים, כולם מגרדים בראשם ומנסים להבין מסמכי אפיון מורכבים...

שורש הבעיה, כפי שאפשר לראות באיור 5, הוא בקשר בין הצוות שיודע מה הוא רוצה (בדוגמה שבאיור זהו מטוס קרב) לבין הצוות שאמור לבנות את החמרה ולכתוב את התכנה שיפעילו את המערכת הזאת. ובצורתה הפשוטה ביותר, הבעיה היא בתקשורת בסיסית בין שני הצוותים האלה: איך הצוות הראשון (הלקוח) מסביר לצוות המתכנן והבונה מה הוא רוצה, כדי שהתוצאה תהיה לשביעות רצונו. ושוב, לב העניין וחלקו הקשה ביותר הוא אפיון ההתנהגות הדינמית הרצויה של המערכת.

## מה לעבדכם הנאמן ולכל זה?

עד שנת 1984 עסקתי בעיקר במחקרים תאורטיים במדעי המחשב. הגעתי לעולם של תכנות מערכות בעקבות ייעוץ של יום בשבוע במשך כמה חודשים בקבוצת האוויוניקה של פרויקט ה"לביא" (אותו מטוס קרב ישראלי שלימים נגנז וייצורו בוטל). הבעיה שם הייתה חמורה מאוד. היה אפשר לדבר עם






מומחי המכ"ם, עם מומחי בקרת הטיסה, עם מומחי החמרה או התכנה או עם מומחי התקשורת המוטסת, ולכל אחד מסוגי המומחים הייתה שפת עבודה אחרת; הם דיברו במונחים שונים זה מזה, והתקשורת ביניהם הייתה איומה למדיי. אם דיברת עם איש המכ"ם, למשל, היית מקבל תיאור מדויק של הדרך שבה המכ"ם מודד את הטווח למטרה. איש התקשורת היה מדבר על הזרמת הנתונים לאורך מטוס, איש החמרה היה מספר על המודולים במקומות השונים במטוס וכו'. אבל כשבא מישהו מבחוץ, כמוני למשל, ושאל שאלה נאיבית כגון "מה

קורה כשאני לוחץ על הכפתור הזה שעל המצערת?", הוא היה מקבל תשובה הלקוחה למשל מעמ' 759 בכרך השני של המסמכים שהיו אמורים לתאר את התנהגות המערכת, ולפיה במקרה כזה "קורה כך וכך...". את התשובה לשאלה השנייה כבר לקחו מכרך אחר, נניח מעמ' 238 אך גם מאיזו פסקה רלוונטית בעמ' 334, וכן הלאה. כלומר שביבי המידע על ההתנהגות הרצויה של המערכת היו מפוזרים במקומות שונים לגמרי, בינות למאות העמודים של המסמכים. היה קשה עד בלתי אפשרי למצוא שם את הידיים ואת הרגליים.

כדאי להדגים את בעיית התיאור ההתנהגותי המלא על מערכת פשוטה בהרבה ממטוס קרב, למשל מערכת שכולנו מכירים היטב – מכשיר ה"כספומט". נוטים לחשוב שזאת מערכת פשוטה, שלא כמטוס ג'מבו או כור גרעיני או מערכת ההפעלה "חלונות" במחשב האישי שלנו. אבל זה לא נכון כלל. אפשר לשאול כל אחד מאתנו שאלות על התנהגות מכשיר הכספומט, שעליהן ניתן תשובה שגויה או לא נדע את התשובה כלל. לדוגמה: נניח שאני מכניס את הכרטיס המגנטי לחריץ המכשיר, מקיש את ארבע הספרות של הקוד, מבקש אלף שקל ועושה את כל הפעולות הדרושות, אבל אחרי שהכרטיס מתחיל לצאת, במקום להוציאו ולחכות ליציאת השטרות אני מושך אותו החוצה ומיד לוחץ על "ביטול". האם תתבטל הפעולה? האם כבר מאוחר מדי והיא לא תתבטל? וכמובן, יש גם האפשרות הנעימה פחות (שאנו מקווים שהיא גם הסבירה פחות) שהחשבון יחויב אבל את הכסף לא נקבל...

זאת דוגמה קטנה מאוד להתנהגות לא ברורה ולא ידועה מראש של מערכת שאנו משתמשים בה יום יום. אם נעשה קפיצה מחשבתית ממכשיר זה, שבמקרה הגרוע ביותר יגרום להפסד כספי שאינו בשמים, לחדר הבקרה של כור גרעיני או לתא הטייס במטוס נוסעים, נראה שהבעיה יכולה להיות בשמים וגם על הארץ...

תכנון נכון ואמין של התנהגות תגובתית של מערכות מורכבות הוא בעיה סבוכה וקשה ביותר. הנה שלושה ציטוטים ממסמך אפיון בן כ־600 עמודים של מפעל כימי אירופי משנות השמונים של המאה העשרים. שלושתם עוסקים בפריט התנהגותי קטן אך קריטי למדיי. הציטוט הראשון והשני לקוחים מפרקים בחלק הראשון של המסמך:

Section 2.7.6: Security

"If the system sends a signal hot then send a message to the operator."

Section 9.3.4: Temperatures

"If the system sends a signal *hot* and T > 600, then send a message to the operator."

הציטוט השלישי – והוא הדובדבן שבקצפת – לקוח מסוף המסמך, שם ריכזו המחברים כמה עניינים התנהגותיים קריטיים לשם סיכום. כדאי לקרוא אותו בעיון:

Summary of critical aspects

"When the temperature is maximum, the system should display a message on the screen, unless no operator is on the site except when T > 600."

המזעזע הוא שגם אם יטרח מתכנתו של החלק הזה של המערכת וימצא בסבך הכרך הענק את שלושת המקומות האלה, קשה להאמין שהתיאורים הלא־עקיבים יביאו לתכנות נכון וצפוי של המערכת ושלא תהיינה הפתעות לא נעימות, בלשון המעטה.

## מחקר בשיטות תִכנות

העיסוק המדעי שלי אינו בתכנות כשהוא לעצמו, אלא במֶטה־תכנות, כלומר בפיתוח התשתית המחשבתית, המדעית־מתמטית והיישומית עבור אלה שעוסקים בתכנון ובתכנות של מערכות כאלה. אני מחפש שיטות טובות, שפות נוחות וכלי עזר חזקים שאפשר לפתח ולספק לאלה שאמורים לבנות מערכות כאלה, כדי שיוכלו לבצע את עבודתם על הצד הטוב ביותר ושהמערכות שייבנו יעשו את מה שרוצים שהן יעשו ולא ייכשלו: שמטוסים וחלליות ותחנות כוח לא יתפוצצו, שקבצים לא יימחקו ומחשבים לא יקרסו וכדומה. המטלה הזאת קשה מאוד, ובמיוחד החלק שמעניין אותי, החלק ההתנהגותי, הדינמי, כפי שניסיתי להדגים למעלה בעניין הכספומט והמפעל הכימי.

מחקר רציני בנושא הזה חייב להידרש לשני הפנים של העניין. הפן הראשון הוא פיתוח שיטות התכנות עצמן, כלומר מציאת הדרכים הטובות ביותר שבהן הצד המורה – למשל ההורה, המפקד או המנהל – יוכל לתת את ההוראות כך שיהיו ברורות ומדויקות. ובעגה המחשבית, פיתוח

הדרכים שבהן המתכנת יתכנת את המחשב. הפן השני חשוב לא פחות, והוא קשור ב"הבנה" של ההוראות, או, במונחים מחשביים, בפיתוח שיטות הרצה: איך הצד המבצע, המחשב למשל, אמור "להבין" את התכנית ולבצע את ההוראות שניתנו על פי שיטת התכנות.

## היסטוריה קצרצרה של שיטות תכנות

בשנות הארבעים של המאה העשרים תוכנתו המחשבים בעזרת שפות מכונה (ראו דוגמה באיור 6). בשפות המכונה הפקודות אינן מובנות לקורא ומדברות במספרים בלבד על מקומות בזיכרון, נתונים המאוחסנים בהם ופקודות המשנות אותם. בהמשך תוכנתו המחשבים בשפות סף מופשטות מעט יותר (בעגה המקצועית הן "עיליות" יותר), ובהן ניתנו שמות נוחים לכל פקודה, למקומות בזיכרון ולערכי הנתונים (ראו איור 7). שפות תכנות עיליות ממש התחילו להתפתח בשנות החמישים. הן היו דומות יותר לאנגלית (ראו איור 8 בעמוד הבא), והיו בהן פקודות כגון אלה:

- **if** condition **do** command **else do** another command
- **while** condition **do** command

6

```
LK 1550 data 193,   2,  32, 210, 255, 169
ME 1560 data   2, 141, 168,   2,  32,  24
HF 1570 data 104, 173, 171,   2, 141, 246
IF 1580 data   2, 173, 172,   2, 141, 247
DB 1590 data   2,  32,   6, 104, 169,   8
OF 1600 data 133, 114, 169,   1, 133, 113
EH 1610 data 169,   0, 141,   0,   8, 141
FH 1620 data 244,   2, 141, 245,   2, 141
NI 1630 data 237,   2, 141, 236,   2, 173
FJ 1640 data 253,   2, 133, 248, 173, 252
MJ 1650 data   2, 133, 247, 173, 249,   2
BL 1660 data 141, 243,   2, 173, 248,   2
ML 1670 data 141, 242,   2, 169, 107, 133
HK 1680 data 140, 169, 241, 133, 139,  32
KO 1690 data 137, 103,  32, 174, 105,  32
HO 1700 data 239, 105,  32,  62, 106, 169
IO 1710 data   0, 141, 237,   2, 168, 177
CA 1720 data 247, 133, 254, 201, 128, 176
HN 1730 data  21, 173, 245,   2, 201,   1
FD 1740 data 208,  32, 169, 146,  32, 232
OP 1750 data 104, 206, 245,   2, 238, 244
MF 1760 data   2,  76, 222, 102, 173, 245
NO 1770 data   2, 201,   1, 240,  11, 169
JC 1780 data  18,  32, 232, 104, 238, 245
AE 1790 data   2, 238, 244,   2, 165, 254
PP 1800 data  10,  74, 201,  34, 208,   6
OG 1810 data  76,  79, 106,  76,  24, 103
BL 1820 data 133, 254,  10,  10, 176,  16
```

7

```
$MOD8253
DSEG
                ORG     20h
Var1            DS      1
STATE           BIT     Var1.0
OUTPUT          BIT     P1.0

CSEG
                ORG     0h
                AJMP    START

                ORG     0Bh
                AJMP    INTERRUPT

START           MOV     IE,#82h
                MOV     TMOD,#01
                MOV     TH0,#FEh
                MOV     TL0,#0Ch
                SETB    STATE
                SETB    TR0

LOOP            NOP
                SJMP    LOOP

INTERRUPT       CLR     TR0
                MOV     TH0,#FEh
                MOV     TL0,#0Ch
                SETB    TR0
                CPL     STATE
                MOV     C,STATE
                MOV     OUTPUT,C
                RETI
                END
```

באמצע שנות השמונים פותחו שיטות תכנות חזותיות־גרפיות, שהרעיון בהן היה לתכנת באמצעות תרשים מדויק לגמרי שמתאר את ההוראות של השינויים הרצויים, המצבים שאליהם המחשב נכנס תוך כדי פעולתו וכו' (ראו איור 9). כל התקדמות כזאת בשיטות ובשפות לתכנות מביאה אתה לא רק נוחות הולכת וגדלה בתהליך התכנות אלא גם אמון הולך וגדל בתוצאות וסיכוי הולך וקטן לשגיאות.

דרך אחת להסתכל על כיוון המחקר שמתואר כאן היא לשאול מה עתיד לבוא עכשיו. שפות עיליות הן אינטואיטיביות למדיי ומנסות לבטא את הדרך שבה מתכנת חושב. שפות גרפיות הן עיליות ואינטואיטיביות עוד יותר, ומנצלות גם את תפיסתנו החזותית. מערכת הראייה המפותחת שלנו מאפשרת לנו לתפוס ביתר קלות מרחב חזותי וכן יחסים בין אלמנטים שונים בתמונה או בתרשים. אך גם בשפות המודרניות ביותר עדיין יש פער ניכר בין הרצוי למצוי, כפי שנראה מיד. מה צופן לנו העתיד בשפות התכנות היא שאלה חשובה מאוד. יש רצון עז לפתח שיטות פשוטות יותר לתיאור ופשוטות יותר לתכנות, כך שעבודתו של המתכנת תהיה קלה ואינטואיטיבית הרבה יותר והסיכוי לשגיאות הרות אסון יקטן עוד יותר.

חשוב להבין שככל ששפת ההוראות, כלומר שפת התכנות, עילית ו"גבוהה" יותר כן פיתוח שיטות לביצוע והרצה מסובך הרבה יותר, וכך כמובן גם אצלנו בני האדם. ככל שההוראות שניתנות לילד מופשטות יותר מבחינת הפרטים ומסתמכות יותר על הבנתו, כושר למידתו ויכולת ההיסק שלו, כן על הילד להיות חכם יותר בביצוע. הוראה לאדם שיזיז את היד כלפי מעלה ואחר כך יוריד אותה קלה יותר לביצוע מהוראה להכין בצק לעוגת שוקולד. ועל כן ככל שאנחנו נותנים למתכנתים או למהנדסי המערכות שפות ושיטות גבוהות יותר, אינטואיטיביות יותר, קרובות יותר לדרך המחשבה שלהם, כן צריך כוח ועצמת חישוב חזקים הרבה יותר, והמתמטיקה והאלגוריתמיקה הנדרשות מתחת לפני השטח כדי לבצע את ההוראות חייבות להיות מורכבות בהרבה.

8

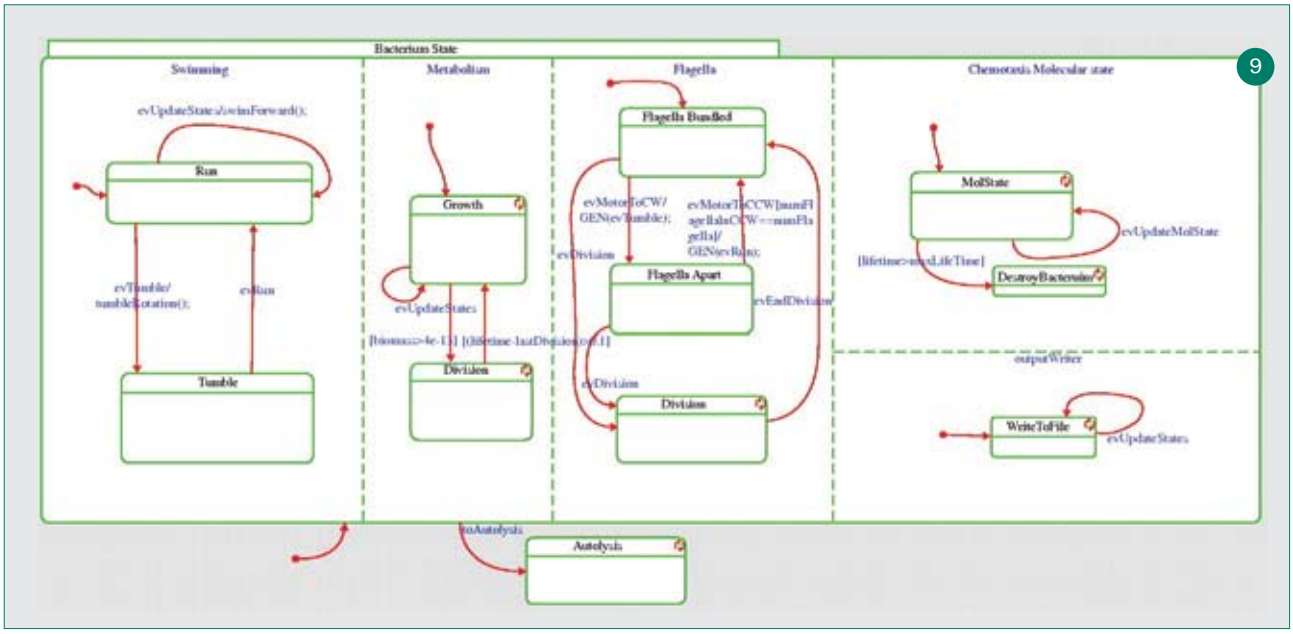
```
42  public class Search implements PagedDataFetcher {
43      protected SearchCriteria searchCriteria;
44      protected int totalNumberOfItems;
45      protected Log log;
46
47      public Search(Log log, SearchCriteria searchCriteria) {
48          assert (searchCriteria != null);
49          this.log = log;
50          this.searchCriteria = searchCriteria;
51          this.totalNumberOfItems = 0;
52      }
53
54      public List<Item> loadMoreItems(EntityManager entityManager, Long userId, int startingRowIndex,
55          int numberOfRows, String sortField, boolean ascending)
56      {
57          FullTextEntityManager ftEm = (FullTextEntityManager)entityManager;
58
59          // get a Lucene Query from the SearchCriteria
60          org.apache.lucene.search.Query luceneQuery = getLuceneQuery(ftEm);
61
62          if (luceneQuery == null) {
63              totalNumberOfItems = 0;
64              return PagedDataProviderBase.NO_ITEMS;
65          }
66
67          FullTextQuery ftq = ftEm.createFullTextQuery(luceneQuery, Wine.class);
68          ftq.setFirstResult(startingRowIndex).setMaxResults(numberOfRows);
69
70          // execute the search
71          List<Item> results = (List<Item>)ftq.getResultList();
72          if (startingRowIndex == 0) {
73              // on first page, save the total number of results
74              totalNumberOfItems = ftq.getResultSize();
75          }
76          return results;
77      }
78
79      protected org.apache.lucene.search.Query getLuceneQuery(FullTextEntityManager ftEm) {
80          org.apache.lucene.search.Query luceneQuery = null;
81          // let the SearchCriteria do the actual Query construction, but it needs
82          // an Analyzer to do so ... get that from HibernateSearch
83          try {
84              luceneQuery = searchCriteria.getLuceneQuery(Wine.DEFAULT_SEARCH_FIELD,
85                                  ftEm.getSearchFactory().getAnalyzer("wine_en"));
86          } catch (ParseException parseExc) {
87              log.error("Failed to parse {0} into Lucene query due to: {1}",
88                  parseExc, searchCriteria.toString(), String.valueOf(parseExc.getMessage()));
89          }
90          return luceneQuery;
91      }
92
93      public String getShortDescription() {
94          return searchCriteria.display();
95      }
96
97      public int getTotalNumberOfItems(EntityManager entityManager, Long userId) {
98          return totalNumberOfItems;
99      }
100 }
```

## החלום

ג'ון בקוס (Backus) היה האיש שהמציא את שפת התכנות Fortran וכתב לה את הקומפיילר (=המהדיר; התכנה שמתרגמת את השפה העילית לשפת מכונה). בכך הוא אפשר לכולנו בפעם הראשונה לתכנת בשפת תכנות עילית. למאמר שהתבקש לכתוב ב־1978 בעקבות זכייתו בפרס טיורינג בחר את הכותרת הזאת: "Can programming be liberated from the von Neumann style?" במאמר זה סקר בקוס שיטת תכנות חדשה ומעניינת מאוד, שהייתה פופולרית בשנות השמונים ועדיין נמצאת בשימוש רחב, שנקראת "תכנות פונקציונלי".

בדיוק שלושים שנה לאחר מכן, בינואר 2008, פרסמתי מאמר בכותרת המשחקת עם הכותרת שלו (ואולי הייתה חצופה מעט...): "Can programming be liberated, period?". במאמר שטחתי לפני הקורא את חלומי בנוגע לשלב הבא בשיטות התכנות. הוא קורא לנסות ולשפר באופן דרמטי את יכולתנו לתכנן מערכות מורכבות מאוד באמינות גבוהה באמצעות שחרור צוות התכנות מהמגבלות שכובלות אותו בשיטות התכנות המקובלות. החלום הוא לקרב את שיטות התכנות והרצת התכנות לדרך שבה אנו בני האדם מחילים באופן טבעי תהליכי "תכנות" על אחרים, כפי שתואר בתחילת המאמר הנוכחי. ההתמודדות עם חלום כזה כרוכה בפיתוח טכניקות חישוב וכלים מתמטיים מסובכים במגוון רחב של נושאי מחקר.

במאמר ההוא מ־2008 תיארתי את שלוש המגבלות המהותיות ביותר, אשר גם עכשיו, בעשור השני של המאה העשרים ואחת, עדיין כובלות את המתכנת, ואשר לא השתנו בהרבה מאז שנות החמישים שבהן התחילו להתפתח שפות




תכנות עיליות. למרות ההתפתחויות העצומות בשיטות התכנות ובשפות שלצדן, עדיין שלוש המגבלות האלה שרירות וקיימות כמעט כמו אז. ואלו הן:

1. הצורך לתת למחשב באופן רשמי ומדויק ארטיפקט פורמלי (התכנית) שיכיל את ההוראות לביצוע (מה על המחשב לעשות ובאיזה סדר).
2. הצורך ליצור בעצם שני ארטיפקטים פורמליים: האחד מכיל את הוראות הביצוע (התכנית) והשני הוא מסמך מדויק נוסף המפרט את ציפיותינו מהוראות האלה (הדרישות).
3. הצורך לחלק את התיאור ההתנהגותי של המערכת על פי החלקים הפיזיים או הקונצפטואליים של המערכת עצמה.

אסביר בקצרה. המגבלה הראשונה נוגעת להיותו של המחשב ביסודו ישות לא חכמה היודעת לעשות בעצמה רק דברים בסיסיים ופרימיטיביים מאוד. נראה שכדי לתכנת מחשב כך שיוכל לבצע את מה שברצוננו לבצע אנו חייבים לתת לו הוראות מפורטות לביצוע, ולכתוב אותן בשבילו בשפה מדויקת ופורמלית.

המגבלה השנייה היא שלא די לתת למחשב הוראות מה עליו לעשות, אלא יש גם להגדיר במדויק את הדרישות מהמערכת. למשל, אם אנחנו נותנים למחשב תכנית לחישוב ממוצע המשכורות בארגון מסוים, מסמך הדרישות יכיל תיאור מתמטי מדויק שאומר שהחישוב אכן יסתיים ולא ירוץ לנצח, ושתוצאתו תהיה תמיד מספר שערכו אכן ממוצע המשכורות של כל העובדים שבקלט שהמחשב מקבל בפורמט מוגדר. אם אנחנו כותבים תכנית בשביל מכשיר כספומט, היינו רוצים שבמסמך הדרישות תופיע גם הקביעה שלעולם לא יחויב חשבון של לקוח בלי שבסוף הפעולה הסכום שבו החשבון מחויב אמנם יצא מן המכשיר בשטרות של כסף. אציין כאן שחלק גדול מהמחקר במדעי המחשב (ובכללו חלק הארי של עבודותיו של אמיר פנואלי, אשר לזכרו מוקדש מאמר זה) עוסק בהעמדתם של התכנית ושל מסמך הדרישות ממנה זה כנגד זה, למשל לצורך הוכחת שקילות בין השניים, שתבטיח שבכל פעם שתורץ התכנית היא באמת תקיים את הדרישות.

המגבלה השלישית היא שבבניית מערכת שיש בה אובייקטים (רכיבים) רבים נראה שהדרך היחידה לתאר את התנהגות המערכת כולה היא לתאר את התנהגותו של כל אחד מהאובייקטים האלה בנפרד, בתקווה שכאשר יופעלו האובייקטים יחדיו יקרה מה שהיינו רוצים שיקרה. אולם לא ברור כלל שזאת הדרך הטבעית ביותר לתאר התנהגות. לעתים קרובות טבעי לתאר התנהגות מסוימת של המערכת שמעורבים בה כמה אובייקטים. מין תסריט שכזה. למשל,

כדי לתאר התנהגות של חדר הרצאות, במקום לתאר את כל אפשרויות הפעולה לכל אחד מהאובייקטים הדינמיים שבחדר (כל אחד מהמשתתפים, המרצה, המחשב, המקרן וכו'), טבעי הרבה יותר לדבר על תסריטי התנהגות שקשורים לכמה אובייקטים (כמו התהליך שבו מאזין שואל את המרצה שאלה, הלה עונה, המאזין ממשיך להקשות וכו', עד שהמאזין בא על סיפוקו או התהליך מסתיים באופן אחר).

אם כן, אלה שלושת הדברים שלדעתי כובלים את עולם התכנות ומגבילים מאוד את יכולתו לפרוץ פריצה גדולה קדימה. החלום הוא לפתח טכניקות תִכנות שישחררו אותנו מהם, אם אפשר.

## האם אפשר להגשים את החלום?

במסגרת הנוכחית קשה לתאר בצורה מספקת פתרונות אפשריים, כיוון שבמאמר כזה אי אפשר להיכנס לפירוט טכני. על כן אקצר ואתן קווים כלליים בלבד למה שאנחנו עושים בכיוון הזה בשנים האחרונות. הספרות המובאת בביבליוגרפיה (חוץ ממאמרו של בקוס) מכילה חומר רב על כך.

ובכן, בקבוצת המחקר שלי במכון ויצמן נעשה מחקר רחב ומקיף בכיוונים שתוארו כאן כבר יותר מתריסר שנים. אפשר לומר בסיפוק שהתוצאות מעודדת, אף על פי שהן עדיין מוגבלות והדרך עוד ארוכה. הגישה שלנו נקראת "תכנות התנהגותי" (behavioral programming), והיא מאפשרת שני דברים כלליים בהתאם לקווים שתוארו למעלה.

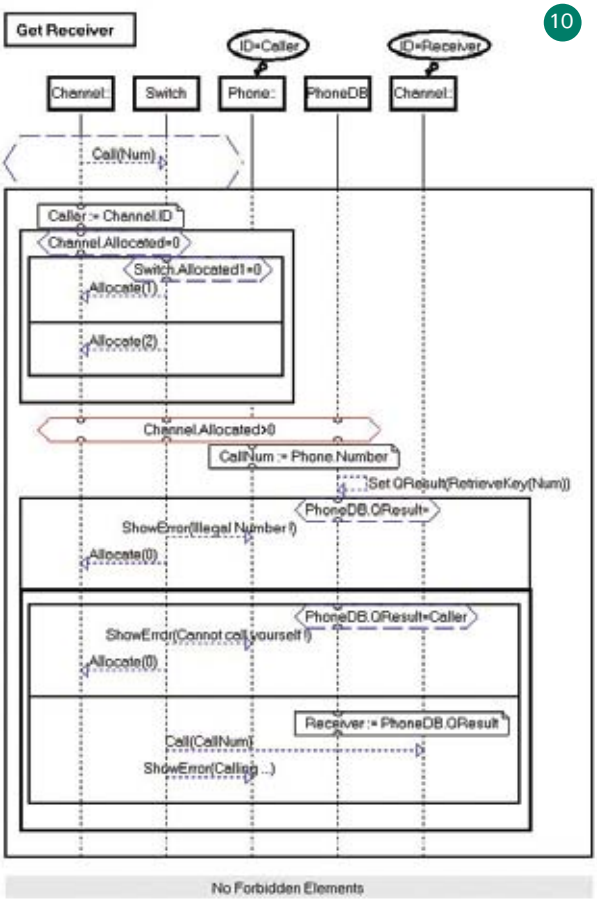
הראשון הוא תהליך תכנות טבעי ואנושי, שנעשה על ידי תיאור תסריטים והדגמות מוחשיות של התנהגויות רצויות, התנהגויות הכרחיות והתנהגויות אסורות בין האובייקטים השונים. תסריטים אלה נקראים בעגה המקצועית "מולטי־מוֹדָליים". לצורך זה פיתחנו שפה חזותית בשם live sequence charts (ראו איור 10), וכן גרסה טקסטואלית, לא חזותית, של הרעיון, המיושמת בשפת התכנות Java. פעולת התכנות עצמה נעשית במה שאני מכנה "נגינה" (play-in) על גבי מנשק גרפי של המערכת שאותה מתכנתים. ה"נגינה" הזאת דומה עד כמה שניתן להפעלה שנפעיל את המכשיר לאחר שיתוכנת. לדוגמה, אם אנו רוצים לתכנת מכשיר כספומט נכין תצוגה גרפית הדומה למכשיר אמתי, על צגו, לחיציו, חריצי הכרטיס והשטרות שלו וכו', והתכנות ייעשה על ידי "נגינה" על המנשק הזה באופן דומה לאופן השימוש במכשיר עצמו לכשיהיה מוכן. נחדיר את הכרטיס לחריץ המתאים, נלחץ על הלחיצים וכו', ותוך כדי ה"נגינה" נודיע למחשב בצורה דומה את מה שהיינו רוצים שהכספומט יעשה בתגובה, ואם תגובה זו חייבת לקרות, אסור שתקרה, אפשרי שתקרה או כדומה.

אוסיף כאן שאחד מנושאי המחקר שעל הפרק נוגע לאפשרות לעשות את ה"נגינה" גם בשפה טבעית – שנוכל לומר למחשב, או להקליד לו, הוראות כגון "כאשר אני לוחץ על הכפתור הזה יקרה כך וכך, אלא אם כן באותו זמן קורה זה וזה" וכדומה.

הדבר הכללי השני שגישת התכנות ההתנהגותי מאפשרת הוא הרצה של כלל התסריטים האלה בשיטה דומה של "נגינה" (play-out). אך הפעם המערכת כבר יודעת מה עליה לעשות, בעוד שב־play-in היא לומדת. הדרך שבה ה־play-out נעשה מעניינת במיוחד כי היא דומה לדרך שבה אדם חי את חייו על פי "ספרי החוקים" הרלוונטיים. זו מעין נגינה קולקטיבית של הסימפוניה כולה. מה שצריך לעשות עושים, מה שאסור לעשות לא עושים ומה שאפשר לעשות בוחרים כך או כך באופן אקראי או לפי פרמטרים הֶיריסטיים.

ניקח לדוגמה סטודנט או סטודנטית לתואר שני במכון ויצמן. אותו אדם נושא על גבו מערכת שלמה של ספרי חוקים שעל פיהם הוא מתנהג: ספר החוקים של מדרשת פיינברג במכון ויצמן, חוקי עיריית רחובות, חוקי המדינה, עשרת הדיברות או הנחיות האסלאם או דוקטרינה אחרת שמנחה את חייו וכו'. בכל פעם שאותו סטודנט מתבקש לעשות משהו (בהנחה שהוא ממושמע), הוא יבצע את מה שמבקשים ממנו, אך רק בתנאי שהדבר איננו סותר איסור או הוראה כלשהם באחד מספרי החוקים. בכל פעולה יעביר אותו אדם בראשו את כל הכללים שבספרי החוקים הרלוונטיים לו, יבצע מה שצריך, לא יבצע מה שאסור ואולי יבצע את מה שאפשרי. בין הדברים האפשריים הוא יבחר בשיטה הֶיריסטית כלשהי. המערכת שלנו מבצעת בדיוק כך את הרצת התכנית.

וכך גישתנו מתמודדת (לפחות באופן חלקי) עם שלושת כבלי התִכנות: היכולת "לנגן" את ההתנהגות הרצויה, במיוחד אם משכללים יכולת זו בעזרת שימוש בשפה




טבעית, משחררת אותנו חלקית מהצורך להכין ארטיפקט פורמלי באופן מפורש; היכולת להכניס לתכנית גם תסריטים אסורים ושאר סוגי מגבלות, ולהריץ הכול יחד, מאפשרת שילוב בין ההוראות למחשב (כמו בתכנית קונוונציונלית) לבין הדרישות עליהן, ובכך משחררת אותנו חלקית מהצורך להכין שני ארטיפקטים שונים ולהוכיח מפורשות שהאחד מתקיים בשני; והיכולת לפרוס את ההתנהגות לחלקים על פי רצונו של המתכנת משחררת אותנו מכבלי הצורך להגדיר את התנהגותם של האובייקטים על פי מבנה המערכת.

## סיכום

ברור שבמאמר קצר ולא טכני שכזה אי אפשר לתאר בצורה מפורטת את המחקר ואת תוצאותיו. אף על פי כן, ברצוני לסיים במה שחייבים לעשות תמיד כשמדברים על אתגר מדעי גדול.

העבודה קשה, קשה מאוד. והדרך להגשמת החלום ארוכה ביותר. פיתוח של שפות ותהליכי תכנות טבעיים ושל תהליכי הרצה שדומים לדרך שבה אנו חיים את חיינו דורש עבודה מדעית מורכבת. אנו משתמשים ברעיונות עמוקים ממדעי המחשב, ממתמטיקה ומהנדסה, תוך שילוב של תהליכי למידה, הבנת שפה טבעית ועוד תחומים רבים. את חלקם אפשר לאמץ משטחים אחרים, לפחות כנקודת יציאה, אך את חלקם צריך להמציא ולפתח בעצמנו.

ובנימה חיובית יותר, אנו כבר משתמשים בהצלחה בשיטות החדשות. למשל, בנינו תָכנות משחקים שבהן כל כלל של המשחק וכל טקטיקה מתוכנתים בנפרד, בדומה לדרך שבה היינו מלמדים ילד לשחק – שלב אחר שלב. תכנתנו כמה מערכות בקרה מורכבות למדי, וכן השתמשנו בגישה החדשה כדי לבנות מודלים ביולוגיים שבהם פיסות ידע שונות, שנתגלו במעבדות שונות ופורסמו במאמרים ביולוגיים שונים, לוקטו ותוכנתו כל אחת בנפרד והן רצות יחדיו.

אני רוצה לחשוב שמה שהצלחנו לעשות עד כה במחקר המנסה לקדם את הגשמת החלום הוא יותר מקצה הקרחון. אך יש להבין שהוא משאיר עבודה רבה מאוד לעתיד, ואת חלקה עדיין אין איש יודע כיצד לעשות.

## ביבליוגרפיה

J. Backus, “Can Programming Be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs”, *Communications of the ACM* 21:8 (1978), 613–641.

W. Damm and D. Harel, “LSCs: Breathing Life into Message Sequence Charts”, *Formal Methods in System Design* 19:1 (2001), 45–80.

D. Harel and R. Marelly, *Come, Let’s Play: Scenario-Based Programming Using LSCs and the Play-Engine*, Springer, Berlin, 2003.

D. Harel, “Can Programming Be Liberated, Period?”, *IEEE Computer* 41:1 (2008), 28–37.

D. Harel, A. Marron and G. Weiss, “Behavioral Programming”, *Communications of the ACM*, to appear, 2012.