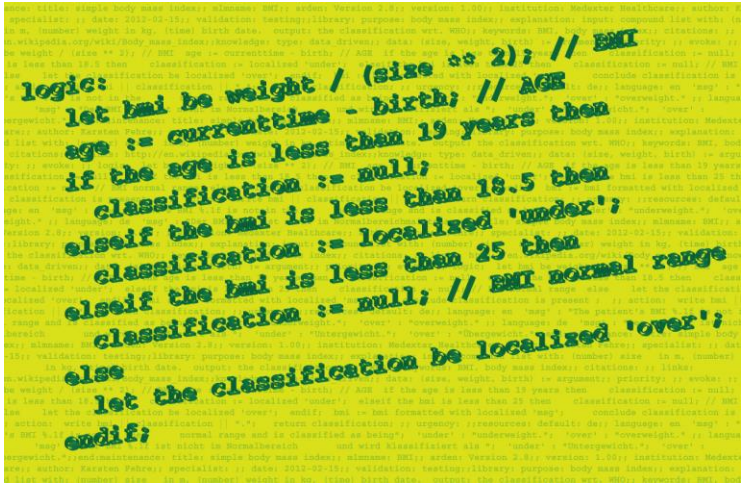
Arden Syntax basics

Educational material, part 3

Medexter Healthcare
Borschkegasse 7/5
A-1090 Vienna

www.medexter.com

www.meduniwien.ac.at/kpa (academic)



Better care, patient safety, and quality assurance by Medexter, Vienna, Austria

Identifying an MLM

- An MLM can be identified by using the following three pieces of information:
  - **Name**, as given in the MLMname slot
  - **Institution**, as given in the institution slot
  - **Version**, as given in the version slot

- Example: The MLM with the following maintenance category

```
maintenance:
   title:       Simplified body mass index;;
   mlmname:     BodyMassIndexSimple;;
   arden:       Version 2.9;;
   version:     1.00;;
   institution: Medexter Healthcare;;
   ...
```

  can be addressed using the following MLM definition in the **data** slot:

```
bmiMLM := MLM 'BodyMassIndexSimple' FROM INSTITUTION "Medexter Healthcare";
```

Note: If there are multiple MLMs with the same name and institution, the MLM with the latest version number is used; also, if a called MLM is from the same institution as the calling MLM, it is no longer necessary to write the institution explicitly.

MEDEXTER Healthcare

Data types – Fundamentals I

- **Null**: Data type that signifies unknown data
- **Boolean**: Includes two truth values (true and false); logical operators use three-valued logic by using null to signify the third value (unknown)

```
TRUE
FALSE
NULL
```

- **Number**: No distinction is made between integer and floating point numbers

```
7
7.34323
```

- **String**: Stream of characters

```
"This is a string constant."
```

Data types – Fundamentals II

- **Time**: Refers to points in time; times before 1800-01-01 are not valid

```
2017-07-12T00:00:12
2017-07-12
```

- **Duration**: Signifies an interval of time

```
19.01 YEARS
3 DAYS 1 HOUR 2 MINUTES 54.6 SECONDS
```

- **Time-of-day**: Refers to points in time that are not directly linked to a specific date

```
23:20:00
```

- **Day-of-week**: Special data type referring to specific days of the week; represented by constants or integer

```
MONDAY   (1)
TUESDAY  (2)
...
SUNDAY   (7)
```

Data types – Fundamentals III

- **List**: An ordered set of elements; each element can be an arbitrary data type (lists cannot contain lists as elements)

```
4, 3, 5
3, TRUE, 5, NULL
,1
()
```

- **Object**: May contain multiple named attributes, each of which may contain any valid data type

```
MedicationDose := OBJECT [Medication, Dose, Status];
dose := NEW MedicationDose WITH "Ampicillin", "500mg", "Active";
// dose refers to an object with the fields medication, dose, status
"Ampicillin" := dose.Medication;
```

Data types – Primary Time

- In addition to its value part, each data value has a **primary time** part and a degree of applicability.
- Primary time represents the value part's time of creation, measurement, examination, etc.
- By default, primary time is `null`.
- Can be accessed using the `time of` operator.

```
2017-03-15T00:00:00 := TIME OF laboratory_result;
```

- Database query results should contain both, the value and the primary time.
  - Might be the time when a blood test was drawn from the patient.
  - Might be the time when a medication order was placed.
  - Which time of a database entry is taken as primary time is left to the Arden Syntax implementer.

Expressions – Fundamentals

- **Statement**: A statement specifies a logical constraint or an action to be performed. All statements except for the last statement in a slot must end with a semicolon (;).

```
LET var1 BE 0; // equal to: var1 := 0;
```

- **Constant**: Any data value that is explicitly represented is called a constant.

```
 TRUE
"this is a string"
```

- **Variable**: A variable is a placeholder for a data value or special constructs (e.g., an event, MLM, message, or destination) and represents this value in any subsequent expressions. An assignment statement is used to assign a value to a variable.

```
LET var1 BE 0;
var2 := MLM 'BodyMassIndexSimple' FROM INSTITUTION "Medexter Healthcare";
var3 := var1 + 1;
```

- **Operator**: An expression may contain an operator and a number of sub-expressions called arguments.

```
3 + 5 //where + is the operator, 3 and 5 are the arguments
```

Statements – Fundamentals I

- **Assignment:** Places the value of an expression into a variable.

```
<variable> := <expression>;
LET <variable> BE <expression>;
```

- **Include:** Includes object, MLM, event, interface, and resource definitions from another MLM

```
mlm2 := MLM 'my_mlm2' FROM INSTITUTION "my institution";
INCLUDE mlm2;
```

Statements – Fundamentals II

- **If-Then:** Permits conditional execution based on the value of an expression.
  - There are three different types of if-then statements:

If-Then:
Block1 is executed if condition is true

```
IF <cond> THEN
   <block1>
ENDIF;
```

If-Then-Else:
Block1 is executed if condition is true, otherwise (if condition is false or anything other than true) block2 is executed

```
IF <cond> THEN
   <block1>
ELSE
    <block2>
ENDIF;
```

If-Then-ElseIf:
Block1 is executed if condition1 is true, if condition2 is true block2 is executed, in all other cases block3 is executed

```
IF <cond1> THEN
   <block1>
ELSEIF <cond2> THEN
   <block2>
ELSE
   <block3>
ENDIF;
```

Statements – Fundamentals III

- **Loops**
  - **While Loop:** Loops as long as the condition is equal to true

```
WHILE <condition> DO
  <block>
ENDDO;
```

  - **For Loop:** Loops over the elements of a list

```
FOR i IN (1 SEQTO 10) DO
  ... // i can be used inside of the loop
ENDDO;
FOR i IN list_of_values DO ... ENDDO;
```

- **Conclude**: Ends execution in the logic slot; if the conclude statement has a single true as argument, the action slot is executed immediately; otherwise the MLM terminates instantly
- **Argument**: If a calling instance passes parameters to the called MLM, the MLM retrieves the parameters via the argument statement
- **Return:** Returns the provided parameter to the calling instance (which may be another MLM or an external instance)

Statements – Fundamentals III – Examples

```
CONCLUDE classification IS PRESENT; // if there is a classification
```

- **Conclude** statement
- "`classification IS PRESENT`" will evaluate to `true`, if the classification variable does not refer to `null`.
- If "`classification IS PRESENT`" evaluates to `true`, the execution of the logic slot stops immediately and the execution of the action slot begins.
- If "`classification IS PRESENT`" evaluates to `false`, the execution of the logic slot also stops immediately but the action slot will not be executed and the evaluation of the MLM terminates.

```
LET patientID BE ARGUMENT;
```

- **Argument** statement which assigns all incoming parameters to the variable `patientID`.

```
RETURN result;
```

- **Return** statement that returns the object `result` to the calling instance (if the MLM is called from another MLM, it will be returned to the calling MLM).

Operators – List Operators I

- **Concatenation:** Appends two lists or turns a single element into a list of length one

```
(4,2) := 4, 2;
(3) := , 3;
```

- **Merge:** Combines two lists, appends a single item to a list, or creates a list from two single items; then sorts the results in chronological order based on the primary times of the elements

```
/* data1 has data value 2 and primary time 2017-01-02T00:00:00, and data2 has data values 1 and 3 and
primary times 2017-01-01T00:00:00 and 2017-01-03T00:00:00 */
(1, 2, 3) := data1 MERGE data2;
NULL := data1 MERGE (data2, 42);
NULL := (4,3) MERGE (2,1); // no primary time -> result is null
```

- **Sort:** Reorganizes a list based on either the element values (keyword data) or the primary times (keyword time); default keyword is data

```
(1, 2, 3, 3) := SORT (1,3,2,3);
(10, 20, 30) := SORT DATA (20, 10, 30);
(30, 20, 10) := REVERSE (SORT DATA (20, 10, 30));
(30, 20, 10) := SORT TIME data3; /* assuming that data3 contains the values 10, 20, 30 with
primary times 2017-01-03T00:00:00, 2017-01-02T00:00:00 and 2017-01-01T00:00:00 */
```

Operators – List Operators II

- **Element:** The element ([…]) operator is used to select one or more elements from a list, based on ordinal position starting at 1 for the first element

```
my_list := 5, 10, 15;
my_list[3] := 20; //contents of my_list are now 5, 10, 20
20 := (10,20,30,40)[2]
```

- **Sublist:** The **SUBLIST ELEMENTS FROM** operator returns a sublist of elements from a designated target list and is similar to the substring operator

```
(1, 2) := SUBLIST 2 ELEMENTS FROM (1, 2, 3, 4, 5)
(1, 2, 3, 4, 5) := SUBLIST 100 ELEMENTS FROM (1, 2, 3, 4, 5)
(4, 5, 6) := SUBLIST 3 ELEMENTS STARTING AT 4 FROM (1, 2, 3, 4, 5, 6, 7)
NULL := SUBLIST 2.3 ELEMENTS FROM (1, 2, 3, 4, 5, 6, 7)
```

- **Clone:** Returns a copy of its argument (mostly used for objects)

```
2017-03-15T15:00:00 := CLONE OF 2017-03-15T15:00:00;
```

MEDEXTER Healthcare

Operators – Logical Operators

- **And:** Performs the logical conjunction of its two arguments; if either argument is `false` (even if the other is not Boolean), the result is `false`; if both arguments are `true`, the result is `true`; otherwise the result is `null`.

```
FALSE := TRUE AND FALSE;
NULL  := TRUE AND NULL;
FALSE := FALSE AND NULL;
```

- **Or:** Performs the logical disjunction of its two arguments; if any argument is `true` the result is `true`; if both arguments are `false`, the result is `false`; otherwise the result is `null`.

```
TRUE  := TRUE OR FALSE;
FALSE := FALSE OR FALSE;
TRUE  := TRUE OR NULL;
NULL  := FALSE OR NULL;
NULL  := FALSE OR 3.4;
```

- **Not:** `True` becomes `false`, `false` becomes `true`, and anything else becomes `null`.

```
TRUE  := NOT FALSE;
NULL  := NOT NULL;
```

Operators – Comparison Operators

- **<,>,<=,=>,=, <>:** These operators have their common meaning; they can handle any data type; if one argument is `null` or types do not match, `null` is returned.
- **Is within … to …:** Checks if the first argument is within the range specified by the second and third argument (inclusive).

```
TRUE  := 3 IS WITHIN 2 TO 5;
FALSE := 3 IS WITHIN 5 TO 2;
```

- **Is within … following …:** Checks if a time is within a defined time period.

```
FALSE := 2017-03-08T00:00:00 IS WITHIN 3 DAYS FOLLOWING 2017-03-10T00:00:00;
```

- **Is in:** Checks membership of the first argument in the second argument (list).

```
FALSE := 2 IS IN (4,5,6);
(FALSE,TRUE) := (3,4) IS IN (4,5,6);
```

- **Is string|number|null etc.:** Returns true if the argument is of the given type.

Operators – Comparison Operators – Example

```
// Classification - the classification is only valid for patients older than 19
    IF age IS LESS THAN 19 YEARS THEN
      classification := NULL;
    ELSEIF result IS LESS THAN 18.5 THEN
      classification := LOCALIZED 'under';
    ELSEIF result IS LESS THAN 25 THEN
      classification := LOCALIZED 'normal';
    ELSE
      classification := LOCALIZED 'over';
    ENDIF;
```

- `"LESS THAN"` is a synonym to `<`
- `"age IS LESS THAN 19 YEARS"` returns `true` if the age is under 19 years

Operators – String Operators I

- **Concatenation:** Converts its arguments into strings and concatenates them afterwards.

```
"null3" := NULL || 3;
"45" := 4 || 5;
"list=(1,2,3)" := "list=" || (1,2,3);
```

- **Formatted with:** Formats a string with a given pattern.

```
"The result was 10.61 mg" := 10.60528 FORMATTED WITH "The result was %.2f mg";
"The date was 2017-01-10" := 2017-01-10T17:25:00 FORMATTED WITH "The date was %.2t";
```

- **Localized:** Returns a string that has been previously defined in the language slot of the MLM's resources category, using a given or the current system's language.

```
"Caution, the patient ..." := LOCALIZED 'msg' BY "en_US";
"Achtung, der Patient ..." := LOCALIZED 'msg' BY "de";
"Caution, the patient ..." := LOCALIZED 'msg'; //use host system language
```

- **AS NUMBER:** Attempts to convert a string or Boolean into a number.
  - `42 := LOCALIZED "42" AS NUMBER;`

Operators – String Operators I – Example I

```
resultMSG := bmi || classification || ".";
```

- The **concatenation operator** concatenates the string `bmi` with the string `classification` and the string `"."`.

```
resultMSG := bmi FORMATTED WITH LOCALIZED 'overw_msg';
```

- `"LOCALIZED 'overw_msg'"` will return the format pattern in the current system language.

```
overw_msg: The patient's BMI %.1f is classified as Overweight.
```

- The **formatted with** operator will then apply this pattern to the calculated BMI.
- The result (a string) is assigned to the string `resultMSG`.
- Assuming the calculated BMI is `29.4324`, and the system language is English, the result of this **formatted with** expression is `"The patient's BMI 29.4 is classified as Overweight."`

Operators – String Operators I – Example II

```
LET THE classification BE LOCALIZED 'over';
```

- The **localized operator** will return the string that is assigned to the term `'over'` in the **resources category**.
- The operator will obtain the string from the **language slot** that matches the current language of the system the engine is running on.
- If there is no language slot for the current system language, the defined default language is used.
- Assuming English as the current system language, the whole statement will assign `"Overweight"` to the field `classification` of the object `result`.

Operators – String Operators II

- **Uppercase, Lowercase:** Converts all characters of a given string to lowercase/uppercase

```
"EXAMPLE STRING" := UPPERCASE "Example String";
"example string" := LOWERCASE "Example String";
```

- **Substring:** Returns a substring of characters from a given string

```
"ab"  := SUBSTRING 2 CHARACTERS FROM "abcdef";
"def" := SUBSTRING 3 CHARACTERS STARTING AT 4 FROM "abcdef";
```

- **Matches pattern:** Determines if a string matches a pattern (similar to LIKE in SQL)

```
TRUE  := "past heart attack" MATCHES PATTERN "%heart%";
FALSE := "past heart attack" MATCHES PATTERN "heart";
```

- **Length:** Returns the length of a given string

```
7 := LENGTH OF "Example";
```

Operators – Arithmetic Operators

- **+, -, *, /, **:** Are used in their common meaning, except one argument is `null` or types do not match

```
2 DAYS := 6 DAYS / 3;
9 := 3 ** 2;
```

- **Cosine, Sine:** Calculates the cosine/sine of its argument

```
1 := COSINE 0;
```

- **Log:** Returns the natural logarithm of its argument

```
0 := LOG 1;
```

- **Abs:** Returns the absolute value of its argument

```
1.5 := ABS (-1.5);
```

- **Ceiling:** Returns the smallest integer greater than or equal to its argument

```
5 := CEILING (4.9);
```

- **Floor:** Returns the largest integer less than or equal to its argument

```
-4 := FLOOR (-3.1);
```

- **Truncate:** Removes any fractional part of a number

```
-1 := TRUNCATE (-1.5);
```

Operators – Arithmetic Operators – Example

```
bmi := weight / (size ** 2); // calculation of BMI
age := CURRENTTIME - birth; // calculation of age
```

- The BMI is calculated by **dividing** the current weight of the patient through the **square** of the current size.
- The result is **assigned** to `bmi`.
- The current age of the patient is calculated by **subtracting** the birthday from the current time.
- The keyword `currenttime` is used to refer to the **current system time.**
- Assuming that the birthday is `1977-12-12` and the current time is `2017-06-12T00:00:00`, after evaluating the statement, the variable age will refer to the duration `39.5 years`.

Operators – Time Operators

- **After, Before:** Addition/subtraction of a duration and a time

```
2017-03-15T00:00:00 := 2 DAYS AFTER 2017-03-13T00:00:00;
2017-03-11T00:00:00 := 2 DAYS BEFORE 2017-03-13T00:00:00;
```

- **Time:** Returns the primary time of the provided parameter

```
2017-03-15T15:00:00 := TIME OF data0;
TIME OF data0 := 2022-03-09T17:20:00;
```

- **Time of day:** Extracts the time-of-day from a given time

```
14:23:17.3 := TIME OF DAY OF 2017-01-03T14:23:17.3;
/* let time of data0 be 2017-01-01T12:00:00 */
12:00:00 := TIME OF DAY OF (TIME OF data0);
```

- **Day of week:** Returns a positive integer from 1 to 7 that represents the day of the week of a specified time

```
7 := DAY OF WEEK OF 2017-08-27T13:20:00; // Sunday
1 := DAY OF WEEK OF NOW; // in case the current day is Monday
```

- **Attime:** Constructs a time value from two time and time-of-day arguments

```
2017-06-20T15:00:00 := NOW ATTIME 15:00:00;
2007-01-01T14:30:00 := TIME OF intuitive_new_millenium ATTIME 14:30:00;
```

Operators – Aggregation Operators I

- **Count:** Returns the number of items of a list.

```
var1 := (12,13,17);
3 := COUNT var1;
```

- **Exist:** Returns `true` if there is at least one non `null` item in a list.

```
TRUE  := EXIST var1;
FALSE := EXIST null;
```

- **Average:** Calculates the average of a number, time, or duration list.

```
14 := AVERAGE var1;
04:10:00 := AVERAGE (03:10:00, 05:10:00);
```

- **Sum:** Calculates the sum of a number or duration list.

```
42 := SUM var1;
7 DAYS := SUM (1 DAY, 6 DAYS);
```

- **Median:** Calculates the median value of a number, time, or duration list.

```
13 := MEDIAN var1;
3 DAYS := MEDIAN (1 HOUR, 3 DAYS, 4 YEARS);
```

Operators – Aggregation Operators II

- **Variance:** Returns the sample variance of a numeric list.

```
2.5 := VARIANCE (12,13,14,15,16);
```

- **Min, Max:** Returns the smallest/largest value in a homogeneous list of an ordered type.

```
14 := MAXIMUM (12,13,14);
```

- **Last, First:** Returns the value at the end/beginning of a list.

```
14 := LAST (12,13,14);
```

- **Latest, Earliest:** Returns the value with the latest/earliest primary time in a list.
- **Seqto:** Generates a list of integers in ascending order.

```
(2,3,4) := 2 SEQTO 4;
(-3,-2,-1) := (-3) SEQTO (-1);
() := 4 SEQTO 2;
```

- **Reverse:** Generates a new list with the elements in reverse order.

```
(3,2,1) := REVERSE (1,2,3);
```

Statements – Call Statements

- **MLM calls:** When the MLM call statement is executed, the current MLM is interrupted, and the named MLM is called; parameters are passed to the named MLM.

```
/* Define find_allergies MLM */
find_allergies := MLM 'find_allergies';
(allergens, reactions) := CALL find_allergies WITH patientID;
```

- **Event calls:** When the event call statement is executed, all the MLMs whose evoke slots refer to the named event are executed; parameters are passed to all evoked MLMs.

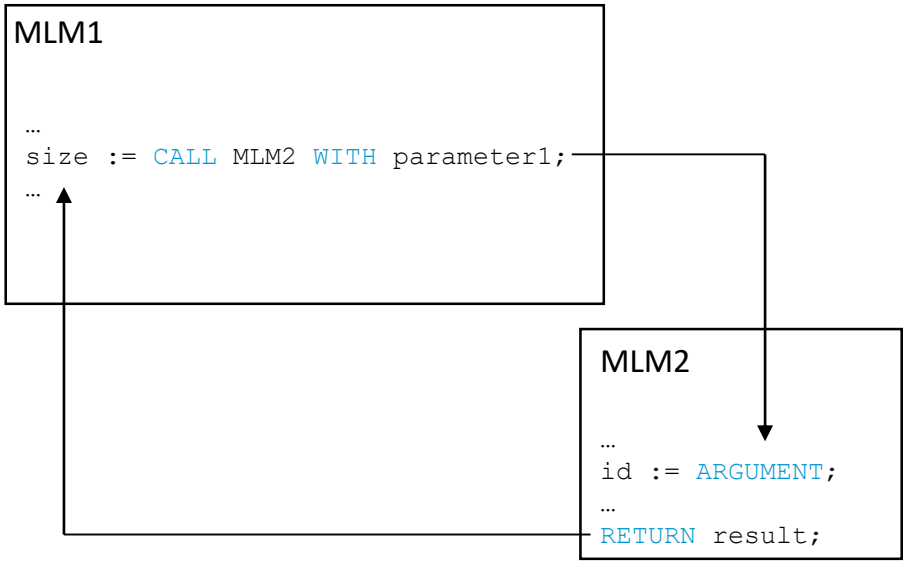
```
allergy_found := EVENT {allergy found};
reactions := CALL allergy_found WITH allergy, patientID;
```

- **Interface calls:** When the interface call statement is executed, the current MLM is interrupted, and the interface is executed; parameters are passed to the interface.

```
/* Define find_allergies external function*/
find_allergies := INTERFACE {\\RuleServer\AllergyRules\my_institution\find_allergies.exe};
(allergens, reactions) := CALL find_allergies WITH patientID;
```

Statements – Call Statements – Nested MLMs

- MLM calls are used to externalize blocks of calculation which may be used by several MLMs or are additionally used in other knowledge bases
- The **call statement** in MLM1 immediately invokes MLM2 (the execution of MLM1 suspends)
- The parameter (`parameter1`) is passed to MLM2 and is accessed using the **argument expression.**
- The passed parameter is assigned to the variable `id`.



- When MLM2 is completed, the result of MLM2 is passed back to MLM1 and assigned to the variable `size` using the return statement.

Statements – Call Statements – Example

```
mlmForReadSize := MLM 'read_Size_MLM';
size := CALL mlmForReadSize WITH patientID;
```

- The **MLM statement** assigns a reference pointing to the MLM `read_Size_MLM`, to the variable `mlmForReadSize`.
- This variable is used in the **call statement** to call the referred MLM.
- The **call statement** passes the content of the variable `patientID` (the patient ID that constitutes the context of the current MLM) to the MLM `read_Size_MLM`.
- The execution of the current MLM is suspended while the called MLM is evaluated.
- The return value of the called MLM is assigned to the variable `size`.

Statements – Triggers

- **Simple Trigger:** A trigger statement specifies an event or a set of events; as soon as any of the events occur, the MLM is triggered; they may only be used in the evoke slot.

```
data:
  penicillin_storage := EVENT {store penicillin order};
  cephalosporin_storage := EVENT {store cephalosporin order};
evoke:
  penicillin_storage OR cephalosporin_storage;
```

- **Delayed Trigger:** Permits the MLM to be triggered some time after an event occurs.

```
MONDAY ATTIME 13:00 AFTER TIME OF penicillin_storage;
```

- **Constant Time Trigger:** Allows the MLM to be triggered at a specific time.

```
2017-01-01T00:00:00
```

- **Periodic Event Trigger:** Allows the MLM to be triggered at specified time intervals after the occurrence of an event.

```
EVERY 2 HOURS FOR 1 DAY STARTING TODAY AT 12:00 AFTER TIME OF event3;
EVERY 1 DAY FOR 14 DAYS STARTING 2017-01-01T00:00:00;
```

Further information

- **The Arden Syntax for Medical Logic Systems Version 2.10**
- HL7 licenses its standards and select IP free of charge
- http://www.hl7.org/implement/standards/product_brief.cfm?product_id=372

Practical Part I