How to use jVE program (September 2002, Pietro Terna)

(jveframe-0.9.7.20.tar.gz)

A "two sides" world description and a consistent program

The first approach to how to use *jVE* (Java Virtual Enterprise), or *jveframe* (a frame used to develop virtual enterprise models based on the Java version of Swarm), introduces the existence of two independent sides in our world description and representation and, in a consistent way, in our program.

Our simulated enterprise has both orders to accomplish – each described by a "recipe" that contains the WD (What to Do) side or the world - and units that perform the different steps of the production process, which represent the DW (which is Doing What) side of the same world.

Units can be within the firm or outside. In the second case: (i) constituting other complex enterprises or (ii) standing alone as small business actors.

It is useful to introduce here a dictionary of our terms:

- a *unit* is a productive structure within or outside our enterprise; a unit is able to perform one or more of the steps required to accomplish an order;
- an *order* is the object representing a good to be produced; an order contains technical information (the recipe describing the production steps) and accounting data;
- a *recipe* is a sequence of steps to be executed to produce a good.

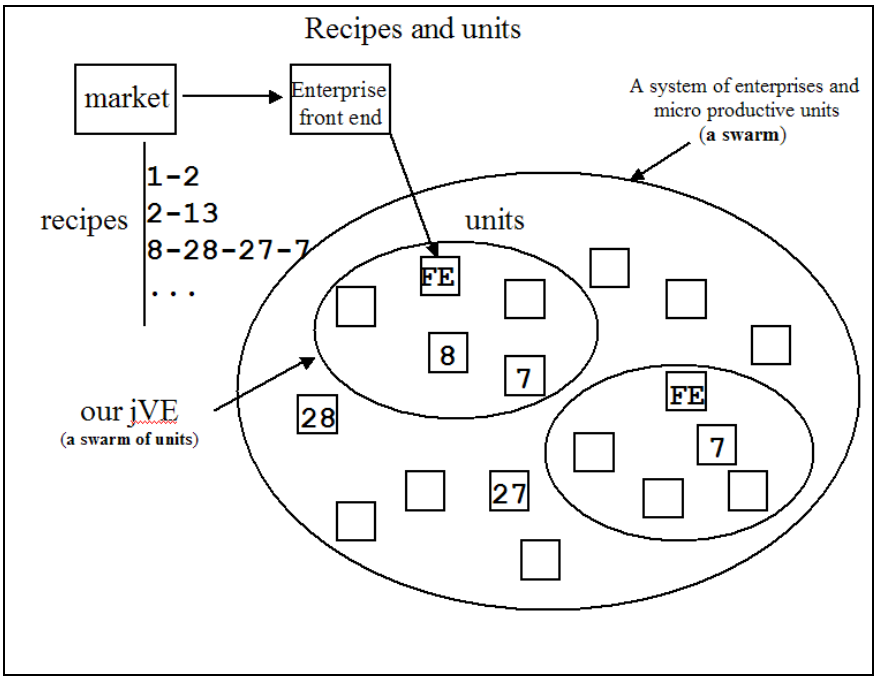

Figure 1. A simplified view of the jVE components.

The core of the model is the clean separation between the order and the units: WD and DW are completely independent, in formalism and in code. So, running the model, we check the consistency of the two sides, as in the actual world, where the output of an enterprise arises

from a complex interaction among products and production tools. As we will see above, recipes can also describe internal parallel production paths, computational steps, batch activities and assembly phases, where the typical procurement problems of a supply chain can be tested (with or without *just in time* requirements).

A simplified view

A simplified view is that of Figure 1.

This is an introductory view of the world, with the recipes written in a simplified way; i.e., as a sequence of steps to be executed without information about the time required by each step. Observing the recipe 8-28-27-7 we can see that the front end (FE) of an enterprise can take in charge the first step, which will be executed by unit 8 (in this simplified version, unit and step numbers are coincident) within the enterprise.

Figure 2 now introduces a more dynamic interpretation of the world we are describing.

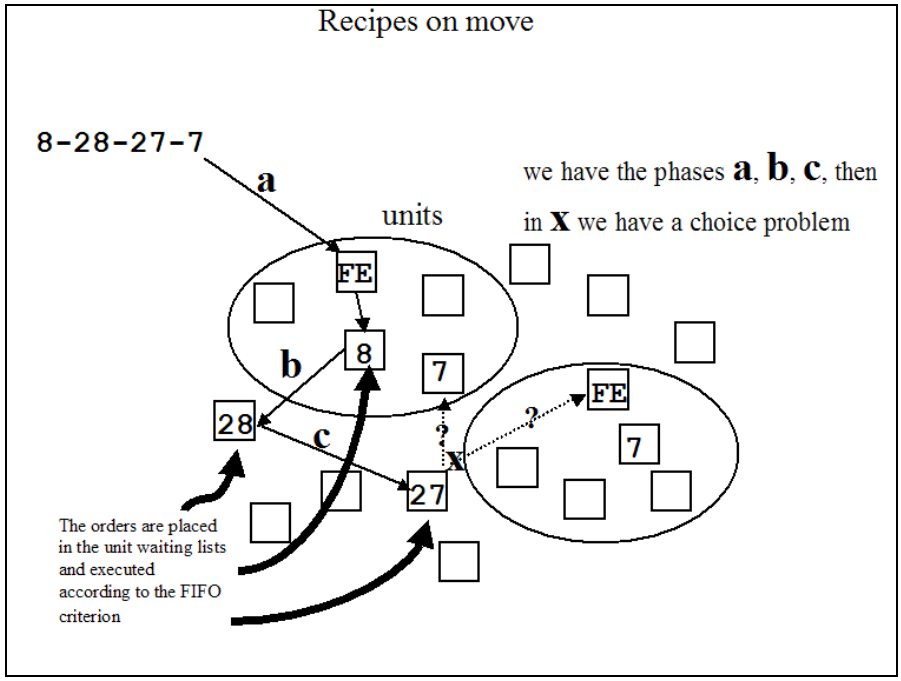

Figure 2. A dynamic view of the jVE components.

We have here three simple phases (*a*, *b*, *c*) in which the order containing the recipe 8-28-27-7 goes from one unit to another; in this sequence, all the needed information is contained in the order: when the activity of a unit (as an example, unit 8) is concluded, the unit asks to the order what is the next step to be performed and then it asks to all the units that is able to execute that task. In this way, the order makes its journey from unit 8 to unit 28 (which is outside the enterprise and can be considered as a simple business unit) and to unit 27 (similar to 28). In the next step, signed with an *x* in Figure 2, we have a choice problem, having to unit able to perform task 7. Below we will introduce a unit criterion properly to deal with this kind of problem in our simulation.

A remark, a little bit more abstract. One of the two units named 7 belongs to another enterprise, so we can imagine of having to open a dialog with the front end of the other enterprise. Anyway we have also to take in consideration the possibility of a direct link with the unit within the other enterprise. The idea of linking together the subunits of more complex enterprises to obtain specific productive results bring directly to the concept of virtual enterprise as an organizational tool: as an example, look at NIIIP project (National Industrial Information Infrastructure Protocols), which as a site at http://niiip01b.npo.org/[2].

A closer look to the WD side

We told that our simulated enterprise has orders to accomplish; the orders are described by the recipes that contains the WD (What to Do) side or the world.

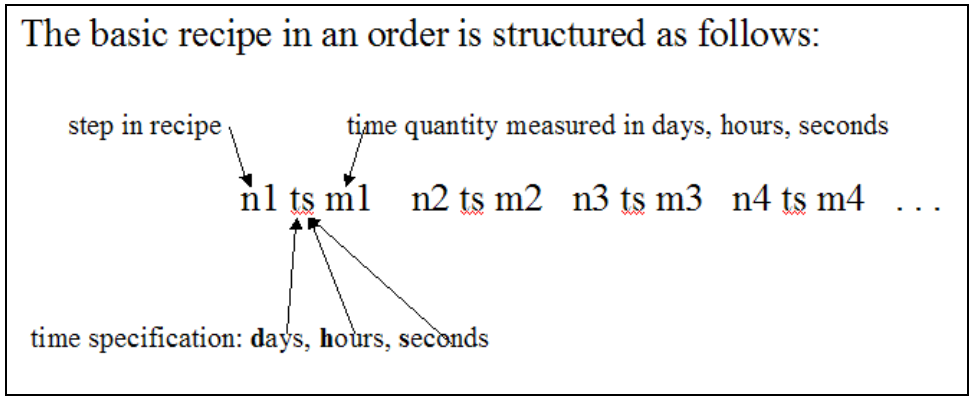

Figure 3. Basic recipe.

The basic recipe in an order is structured as shown in Figure 3.

[1] In the site we can read that: "The NIIIP Consortium consists of a group of leading United States information technology suppliers, industrial manufacturing end users, academic, and standards organizations with a common interest in developing an information infrastructure architecture to enable organizations to operate as "Virtual Enterprises". Virtual Enterprises are teams, consortia or alliances of companies formed to exploit business opportunities that can not be addressed by a single organization."
"The NIIIP Consortium is national in scope and its members bring a wealth of experience and technology to support Virtual Enterprises. Together with the Federal Government, they share costs and skills to create the necessary infrastructure to support Virtual Enterprises across the United States. The NIIIP Consortium has entered into a series of cooperative agreements with the Federal Government and associated agencies to develop, demonstrate, and prototype industrial «Virtual Enterprises»."

[2] In the site we can read that: "The NIIIP Consortium consists of a group of leading United States information technology suppliers, industrial manufacturing end users, academic, and standards organizations with a common interest in developing an information infrastructure architecture to enable organizations to operate as "Virtual Enterprises". Virtual Enterprises are teams, consortia or alliances of companies formed to exploit business opportunities that can not be addressed by a single organization."
"The NIIIP Consortium is national in scope and its members bring a wealth of experience and technology to support Virtual Enterprises. Together with the Federal Government, they share costs and skills to create the necessary infrastructure to support Virtual Enterprises across the United States. The NIIIP Consortium has entered into a series of cooperative agreements with the Federal Government and associated agencies to develop, demonstrate, and prototype industrial «Virtual Enterprises»."

Here we have a sequence of steps followed by a time specification and by a time quantity: step *n*1 requires *m*1 units of time (days, hours o seconds, following *ts* choice). Time quantities are integer numbers.

Obviously we have productions requiring less than one second to make a certain step, but in these cases it is not realistic to think about processes concerning separately single pieces. A realistic view is that of considering the production as batches of pieces. We have to kind of batches in our world: sequential batches and stand alone batches.

A sequential batch process – as reported in Figure 4 – deals simultaneously with a lot of orders, being one of the steps of a recipe. We have to imagine a productive process that is separately managed for each order, but that for certain steps requires an activity referred to a group of orders to be processed together: this is a sequential batch, formally expressed as in Figure 4.

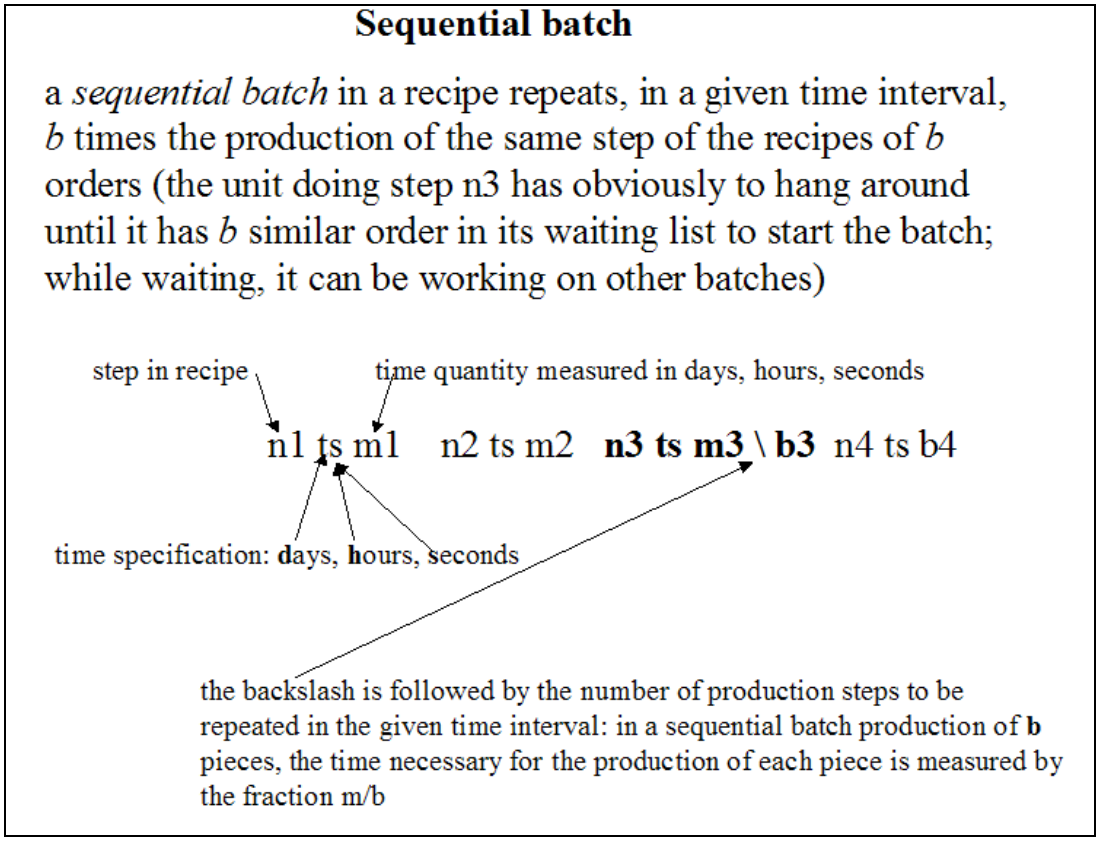

Figure 5. Sequential batch.

A stand alone batch process, described in Figure 6, is similar to a sequential one (we use here "/" instead of "\"), but it is not included in a recipe with other steps: it is the only step of a recipe describing a process considered as a black box: imagine in this case external procurements that our enterprise is ordering in batches of a large dimension, requiring a time delay to be accomplished. In the just in time perspective, the determination of the time point in which to start a stand alone batch order is very important, properly due to the time delay necessary to produce the whole bunch.

Now it would be the time of introducing the procurements, which are key elements in running the virtual enterprise, but we have to know something more about units and "end units", so we go to the DW side; then we will come back to WD.

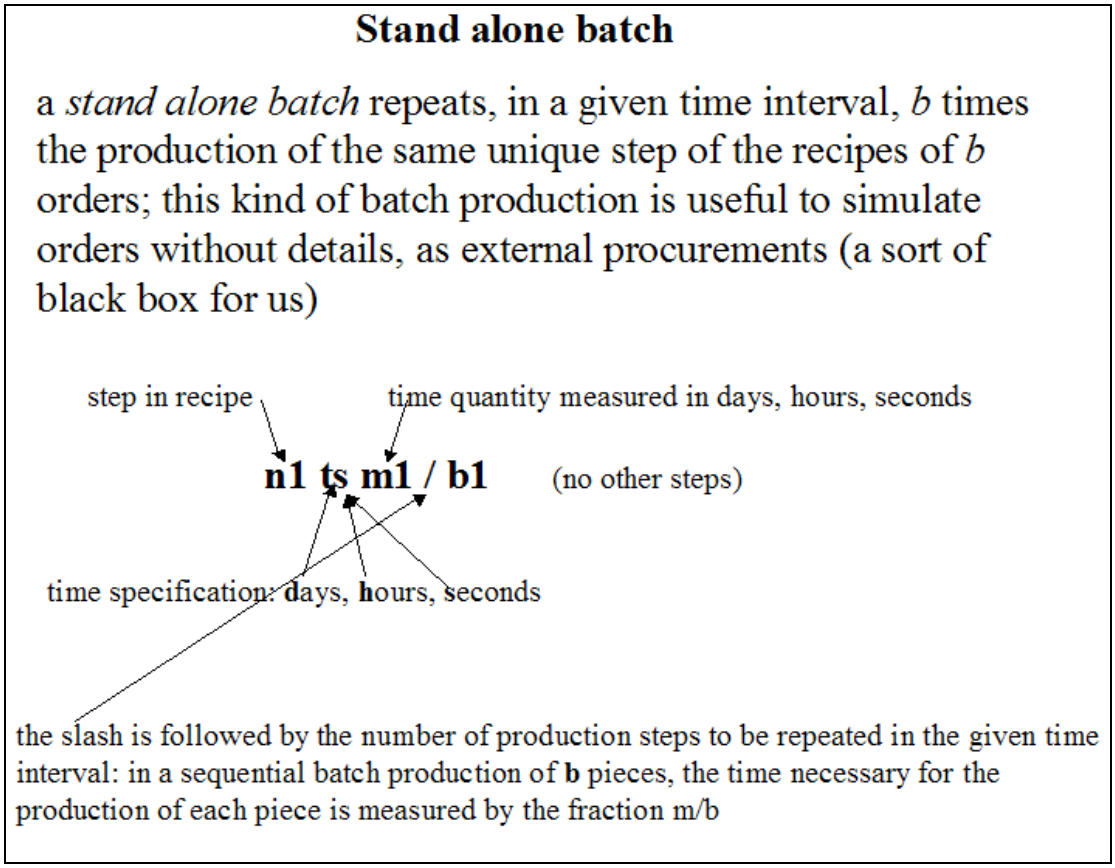

Figure 6. Stand alone batch.

A closer look to the DW side

@@@The DW (which is Doing What) side of the same world is related to units and to "end units".

A unit is the elementary production cell able to accomplish one or more kind of step of a recipe; steps in recipes are identified by number, as we have seen; units too express the steps that they are able to accomplish as numbers.

Simple units, which are able to deal only with one kind of steps, are easily described using the file unitData/unitBasicData.txt that contains the information of Figure 7.

The first line is mandatory, written as is, to force the user to pay attention to the content of the file. Then we have lines reporting: (i) the numbers of the unit (lines have not to be ordered by unit number); (ii) the specific step that the unit is able to do (several units can be able to perform the same step): (iii)

Simple unit data are reported in a text file
(unitData/unitBasicData.txt)

unit_#	useWarehouse	prod.phase_#	fixed_costs	variable_costs
1	1	11	12	1
2	1	0	0	0
3	1	3	15	2
4	1	0	0	0
5	1	51	12	2
6	1	6	11	20
7	1	12	23	1
8	1	8	22	11
9	1	13	7	12
10	1	18	40	7
11	1	11	5	1

Figure 7. Simple units ???.

Complex units

Figure 8. Complex units ???.

End units

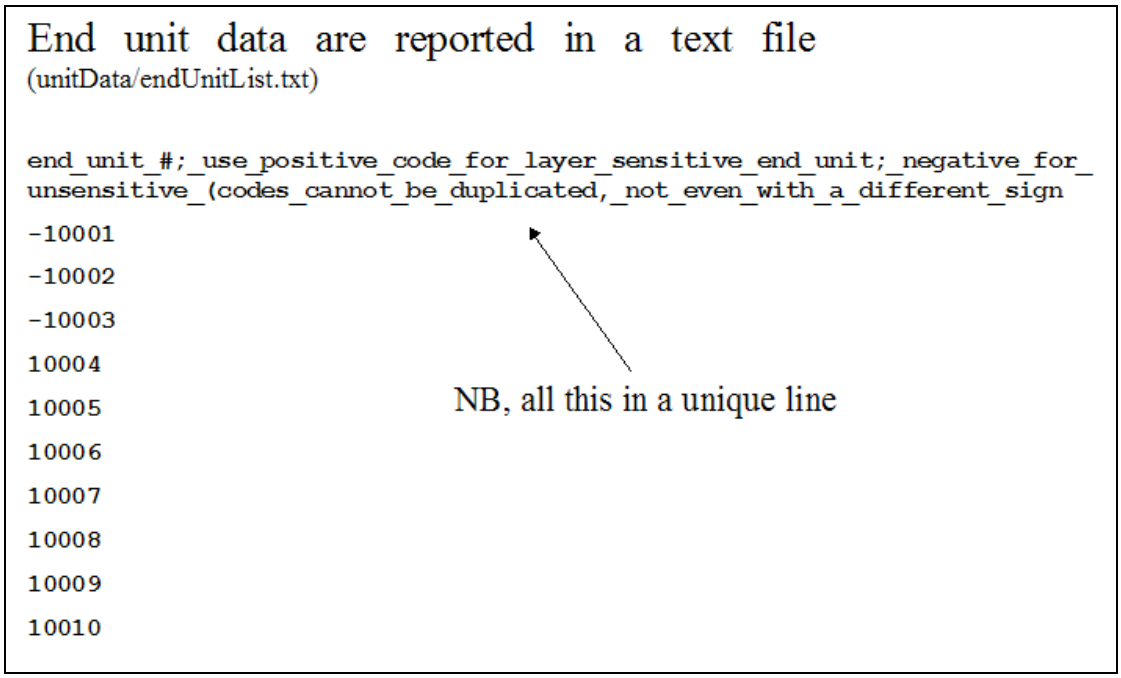

Figure 9. End units ???.

To be implemented: units locking resources for other units

Warehouses

Newses

Newly back to the WD side

External (more human readable) and intermediate format of recipes (the internal one, apparently poor in details, can be examined looking at the comments in Order.java file); anyway we write recipes in external code; the translation mechanism from external to intermediate code is contained in OrderDistiller class; from intermediate to internal, in Order class

Procurements, AND; layers, to be written (AND is also not yet implemented)

OR processes in WD side

We can insert an 'or' choice in a recipe using the format introduced in Figure za. In the example reported here, after step 1, we can have the sequence with the two steps n2 n3 or that with the unique n22, then the execution of the recipe continues with the step n4. The number of branches into the or sequence has no limits.

What branch to choose into the or? We have to look at the orCriterion variable, which is set either via the probe of the model or into the jveframe.scm file.

If orCriterion is:
== 0, all branches are executed in sequence (useful mainly for test purposes);
== 1, the first branch is chosen;
== 2, the second branch is chosen;
== 3, the choice of the branch in made randomly (a good simulated solution if we have to balance the loading of different production subprocesses;
== 4, we choose the branch whose first step has the shortest waiting list;
== 5, we use the result of a computational step to choose the branch (see below, the paragraphs "Computational capabilities and memory matrixes" and "Computational capabilities and 'OR' sequences").

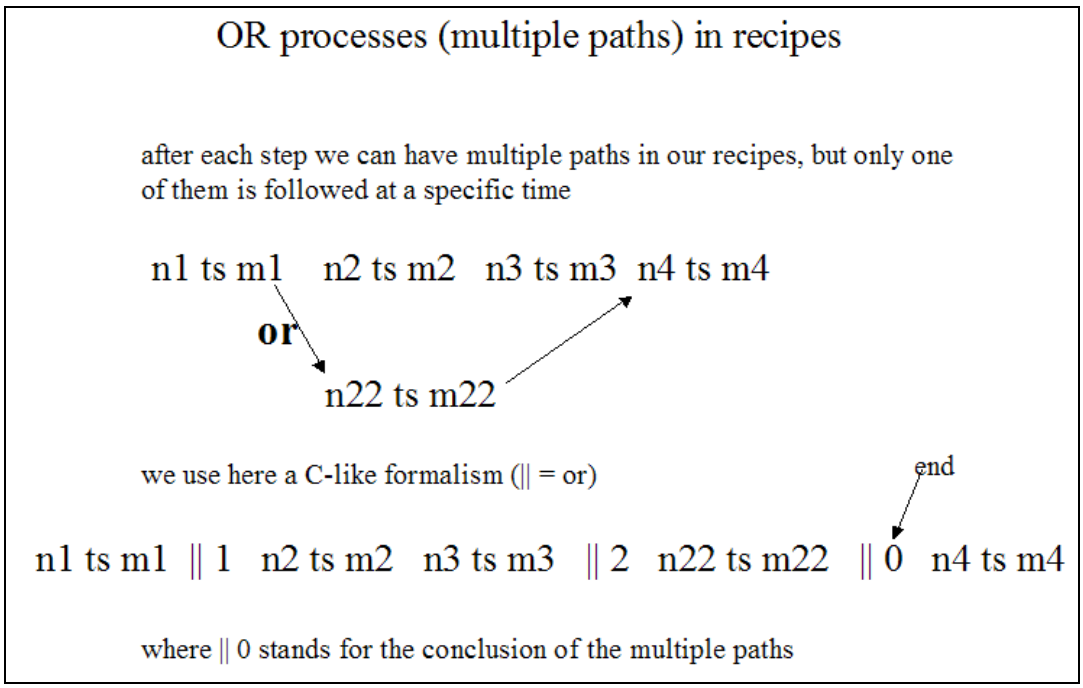

Figure za. An 'or' process, with its branches (|| 1 and || 2)

An example of or sequence is the following, containing also a procurement process (see above; ... *not yet written*) in I one of the or branches:

10 s 3 c 1997 1 2 12 s 0 || 1 11 s 2 p 1 101 10 s 1 9 s 2

|| 2 c 1995 1 0 1 s 0 14 s 3 || 0 6 s 2

where || 1 and || 2 are two nodes opening two branches of the of the 'or' sequence and || 0 concludes them; in the first branch we can identify the simple procurement sequence 'p 1 101 10 s 1'.

The 'or' sequence are managed by the code of jVE in a simple way: al the steps of the discarded branches are immediately signed as executed, then execution proceed in sequence, avoiding those steps fictitiously executed.

Computational capabilities and memory matrixes

jVE has computational capabilities that can be associated to each step of a recipe. To use this feature of the program it is necessary to understand Java language, as we have to modify[3] the ComputationalAssembler.java file (which inherits its default methods from the class ComputationalAssemblerBasic). The goal of the computational capabilities is that of dealing with forecasting, evaluations, auctions to chose procurements, …

Computations use data contained in memory matrixes created following both the totalMemoryMatrixNumber of the model probe (this parameter, stating how many matrixes we are creating, can also be set via the jveframe.scm file) and the contents of the file unitData/memoryMatrixes.txt shown in Figure xa. Memory matrixes use layers in a completely automated way; we can prevent them from using layers setting their number as negative in each specific declaration into the file unitData/memoryMatrixes.txt. In the example reported here, the second matrix (numbered 1, being 0 the number of the first one) in insensitive to layers

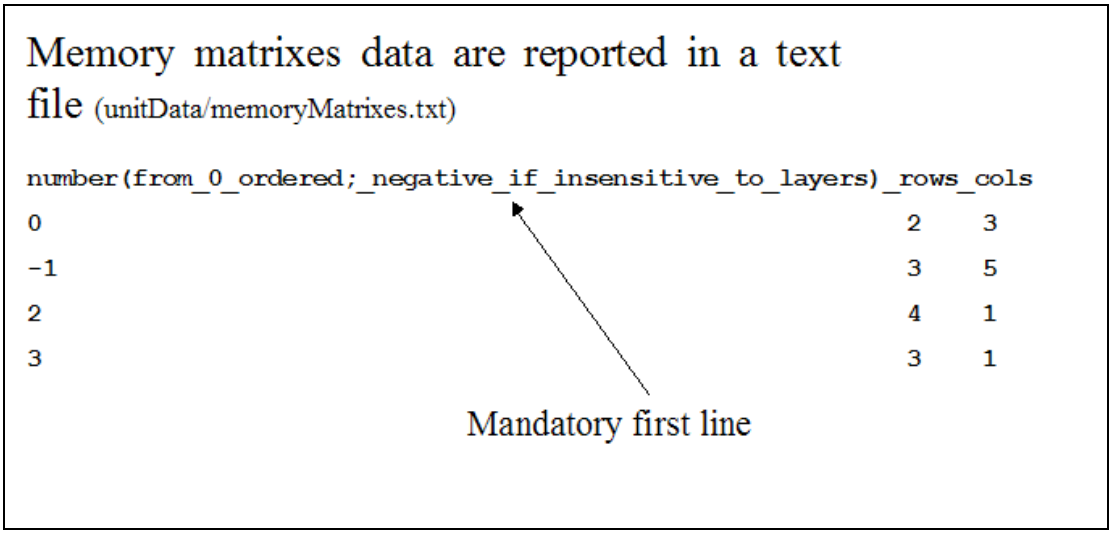

Figure xa. Memory matrixes declarations.

Examples of recipes containing computational steps as reported in Figure xb; obviously, to understand the meaning and the behaviour of a computation it is necessary to consider together both the sequence of the events emerging from the various orders in execution (with

[3] We have not to modify the basic file (ComputationalAssemblerBasic.java), which is included in the src/ folder. Instead, we have to copy in the main folder of the program, from src/, the file ComputationalAssembler.java.
The 'make run' command uses the classes contained in lib/jveframe.jar (which are those contained in src/), but the classes in ./ override those in jveframe.jar.
ComputationalAssembler.java contains no method; we simply add methods, following the examples reported below and using as a guide the full code or the methods reported in ComputationalAssemblerBasic.java. New methods are automatically used by the che ckingComputationsAndFreeingOrders() method of ComputationalAssembler class (which inherits it form its parent class): the tick used to convert the numerical code of the computational steps into a recognized method reference is based upon the java reflection mechanism. To understand the trick, looks at the following lines in ComputationalAssemblerBasic.java code:

```
Class c = this.getClass();
Method m = c.getMethod("c"+(-1*t),null);
m.invoke(this, null);
```

the related operations interesting the memory matrixes) and the content of the Java code of the computational operator itself.

As seen above (… *not yet written*), it is important here to consider both the external (human readable) format of the recipes and the intermediate one, always human readable, but semi-translated. To see the internal code (apparently poor in details) you can have a look to the comment lines in Order.java file. Code number of computational steps are accepted in the range 1001-1999.

The format of a computation is: 'c *code n* m_1 … m_n' where 'c' is mandatory, '*code*' is the code of the computation, '*n*' is the number of matrixes to be used and 'm_1 … m_n' are the numbers of those matrixes, as reported in the file unitData/memoryMatrixes.txt (Figure xa).

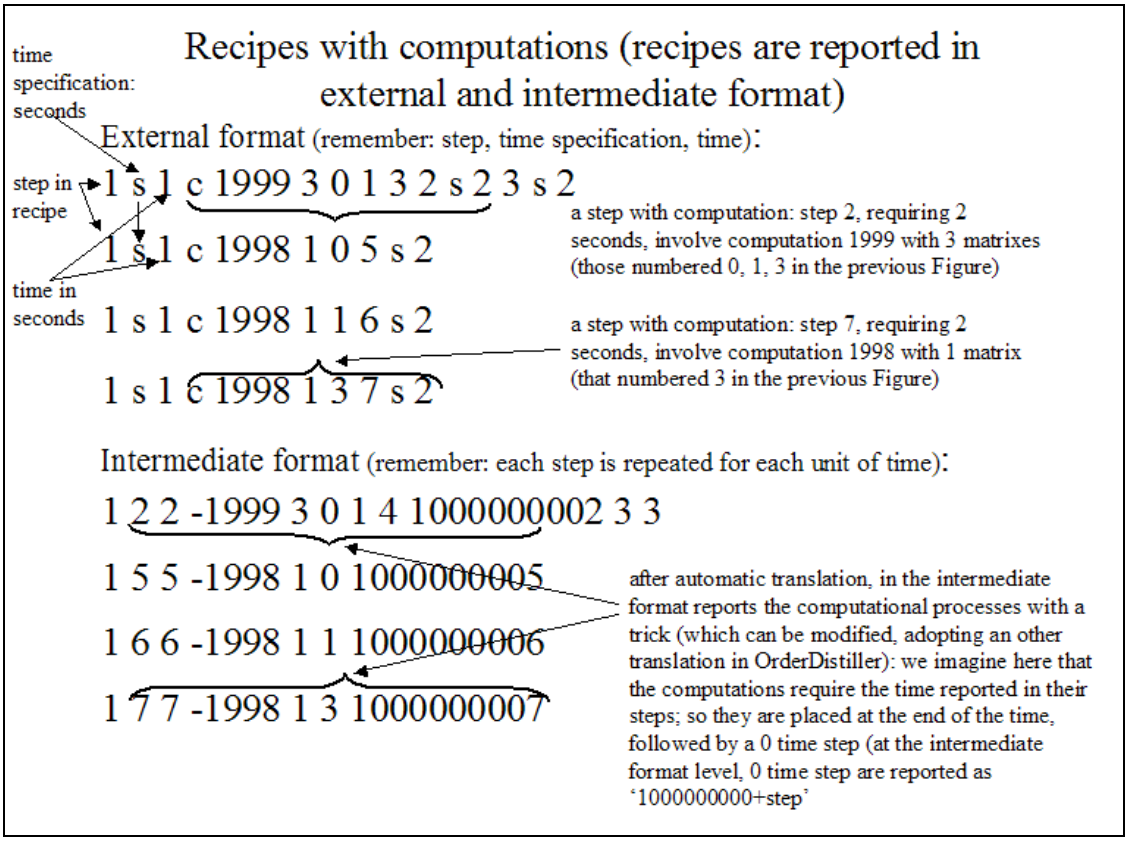

Figure xb. The format of the computational processes.

We introduce some recipes (Figure xb) with computations as a complete example, to explain the dynamics of the events and the Java code related to them. To prepare other computational tools, we have to add lines similar to those introduced below (Figure xc1 and xc2) into the ComputationalAssembler class (ComputationalAssembler.java, as explained in the note above).

In Figure xb we can see how computational codes are represented following their external and intermediate formats (anyway, remember that we write recipes in external code). Pay attention: computational codes at the intermediate format representation level are reported as negative, following the internal convention of jVE, where all the codes related to production steps are positive, while numbers bearing special meanings are negative.

The Java Swarm codes, extracted from ComputationalAssembler.java and reported in Figure xc1 and xc2, interact with the recipes of Figure xb.

When an order with recipe '1 s 1 c 1998 1 0 5 s 2' is executed, at the end of the two units of time required by step 5, matrix 0 is interested by a writing operation in position (0,0) in the proper layer (determined by the level of the order containing the recipe); if the order contains recipe '1 s 1 c 1998 1 0 6 s 2' the writing operation, at the end of step 6, concerns matrix 1 at position (0,0) without layer, being that matrix insensitive to layers by construction; if the order contains recipe '1 s 1 c 1998 1 0 7 s 2', the writing operation, at the end of step 7, concerns matrix 3 at position (0,0) in the proper layer, as above. In the Java code of Figure xc1 we can see those operation made upon mm0 matrix (but we can use any name) related to the actual matrix via the getMemoryMatrixAddress method; the setValue method set the 1.0 value at (0,0). If the matrix is insensitive to layers, the layer value set in this method is disregarded. Finally, the computational step is set as 'done'[4].

The Java Swarm code used by the recipes with computation of this example

```
/** computational operations with code -1998 (a code for the checking
   *  phase of the program
   *
   *  this computational code place a number in position 0,0 of the
   *  unique received matrix and set the status to done
   */
  public void c1998(){

            mm0=(MemoryMatrix) pendingComputationalSpecificationSet.
               getMemoryMatrixAddress(0);
            layer=pendingComputationalSpecificationSet.
               getOrderLayer();

            mm0.setValue(layer,0,0,1.0);

            done=true;
  } // end c1998
```

Figure xc1. The Java Swarm code … (simplified eliminating a control statement related to the consistence of the declared number of matrixes with the internal one).

When an order with recipe '1 s 1 c 1999 3 0 1 3 2 s 2 3 s 2' is executed, at the end of the two units of time required by step 2, matrix 0, 1 and 3 are interested by a check operation to verify if positions (0,0) are empty at the proper layer; if not empty, the 'c 1999' set those positions (at those layers) empty and finally set as 'done'[5] the computational step. Into the code of this

[4] If the Java code related to a computational method does not set 'done' boolean variable to 'true' the order is not freed and does not proceed to its successive recipe steps; the computational step will be repeated in any simulation cycle, until 'done' variable becomes 'true'.

[5] See previous note.

example, matrixes mm0, mm1 and mm2 are linked to actual matrixes 0, 1, 3 (the internal name are completely free).

The effect of those four recipes (OrderGenerator, while testing the program, if totalEndUnitNumber > 0, launches those recipes at random) is the following: the recipe containing the code ‘c 1999’ cannot proceed in step 2 if does not exist the effects of one of each of the recipes containing codes ‘c 1998’ (produced when those recipes are executed at least at step 5 or 6 or 7 deepness). When the recipe containing the code ‘c 1999’ finally proceeds to its successive step, the effects or the “used” recipes is eliminated and must be reviewed by other similar orders.

The Java Swarm code used by the recipes with computation of this example

```
/** computational operations with code -1999 (a code for the checking
  * phase of the program
  *
  * this computational code verifies position 0,0 of the three
  * received matrixes; only if these positions are all not empty
  * the code empties them and set the status to done
  */
 public void c1999(){
        mm0=(MemoryMatrix) pendingComputationalSpecificationSet.
          getMemoryMatrixAddress(0);
        mm1=(MemoryMatrix) pendingComputationalSpecificationSet.
          getMemoryMatrixAddress(1);
        mm2=(MemoryMatrix) pendingComputationalSpecificationSet.
          getMemoryMatrixAddress(2);
        layer=pendingComputationalSpecificationSet.
          getOrderLayer();

        if(! (mm0.getEmpty(layer,0,0) || mm1.getEmpty(layer,0,0)
                        || mm2.getEmpty(layer,0,0) ) )
          {
                mm0.setEmpty(layer,0,0);
                mm1.setEmpty(layer,0,0);
                mm2.setEmpty(layer,0,0);
                done=true;
          }
} // end c1999
```

Figure xc2. The Java Swarm code … (simplified eliminating a control statement related to the consistence of the declared number of matrixes with the internal one.

Method accepted by MemoryMatix instances are setValue, getValue, setEmpty, getEmpty (returning true or false).

The syntax is (leave ‘layer’ as is and set the proper value of the variable as shown in the examples):

- `setValue(layer, (int) row, (int) col, (double) value)` or `setValue(layer, (int) row, (int) col, (float) value)`
- `(float) getValue(layer, (int) row, (int) col)`
- `setEmpty(layer, (int) row, (int) col)`
- `(boolean) getEmpty(layer, (int) row, (int) col)`

where the setEmpty and the getEmpty methods are useful to manage conditional situation; obviously, to set not empty a position of a matrix, we simply pur a value in it; getEmpty returns 'true' if no value is found, other wise it return 'false'.

Computational capabilities and 'OR' sequences

If orCriterion == 5 (se above "OR processes in WD side") computational results are also useful to chose what branch to execute in an or process.

We choose the branch whose number is stored in (x,0) position in the memoryMatrix designated by orMemoryMatrix in the probe of the model or in the file jveframe.scm; the matrix may be sensitive or insensitive to layers. Range of the branch number: from 1 to the number of branches.

x is 0 if the first node in 'or' sequence is numbered 1; is *kk* if the first node is numbered 10*kk* with *kk* 00 to 99. If orCriterion is not equal to 5, the codes 10*kk* are used as 1.

A computational sequence can be included in an 'or' branch with a great flexibility of computational processes[6].

`NB riferimenti in ppt per Seattle; + in articolo SI per il Mulino, soprattutto per l'articolo, non per l'how to`

NB NB cambiare l'ordine dei parametri nella probe del model mettendo in fondo quelli che intessano solo le fasi di test NB spiegare l'uso del distiller di B&B&B rispondendo a "ma come si passano le ricette"

Presentare le probe dell'observer e del model

NB How to get jVE

[6] This aspect is strategic for the development of jVE with the capabilities of simulating both the financial side of the enterprise and the enterprise information system.