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

(jveframe-0.9.7.10.tar.gz)
INCOMPLETE DRAFT, please do no quote

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.

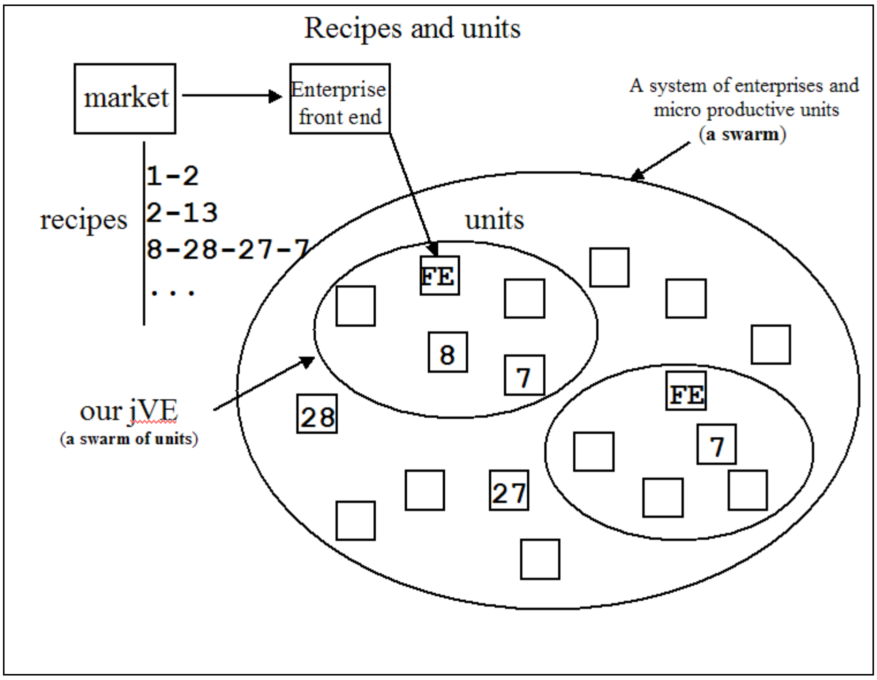


Figure 1. A simplified view of the jVE components.

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.

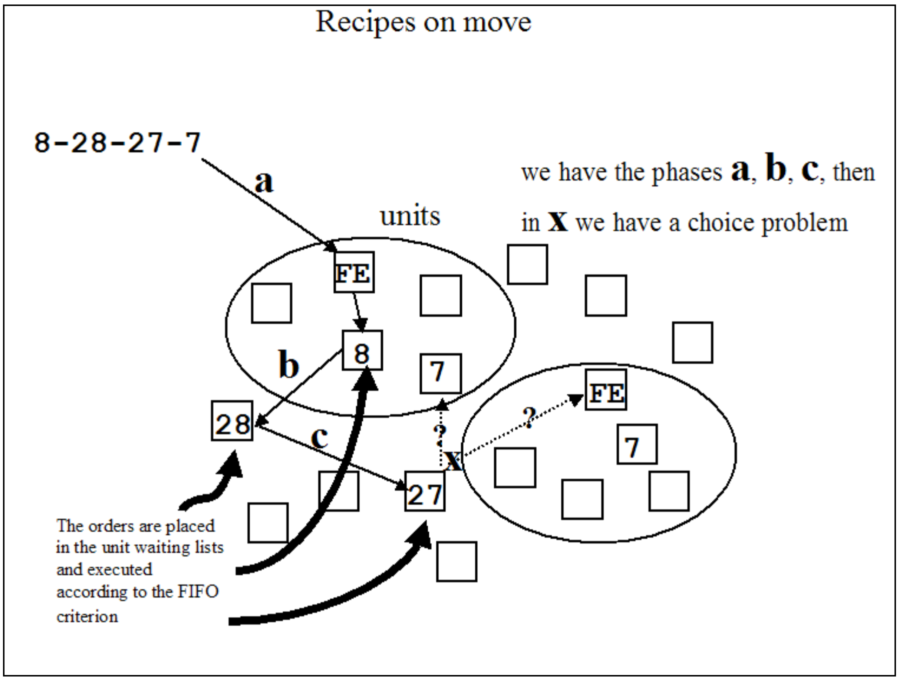

Recipes on move
8-28-27-7
a
units
we have the phases a, b, c, then
in x we have a choice problem
FE
8
7
b
28
c
27
?
x
?
FE
7
The orders are placed in the unit waiting lists and executed according to the FIFO criterion

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.

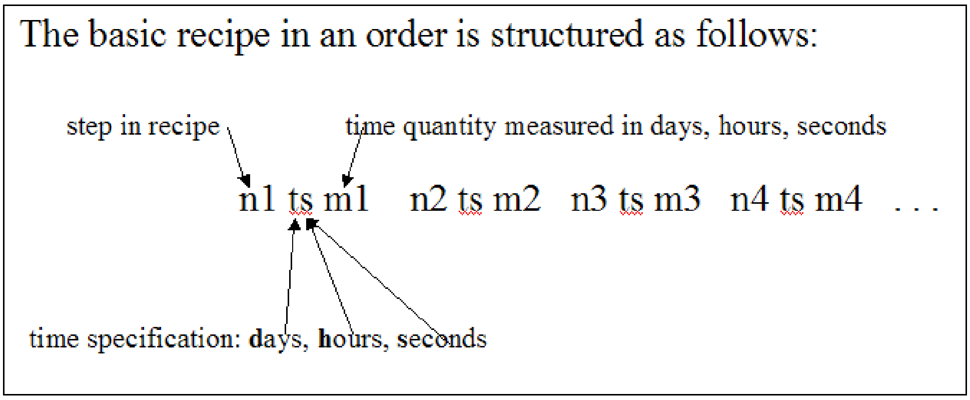

The basic recipe in an order is structured as follows:
step in recipe
time quantity measured in days, hours, seconds
n1 ts m1 n2 ts m2 n3 ts m3 n4 ts m4 . . .
time specification: days, hours, seconds

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».”

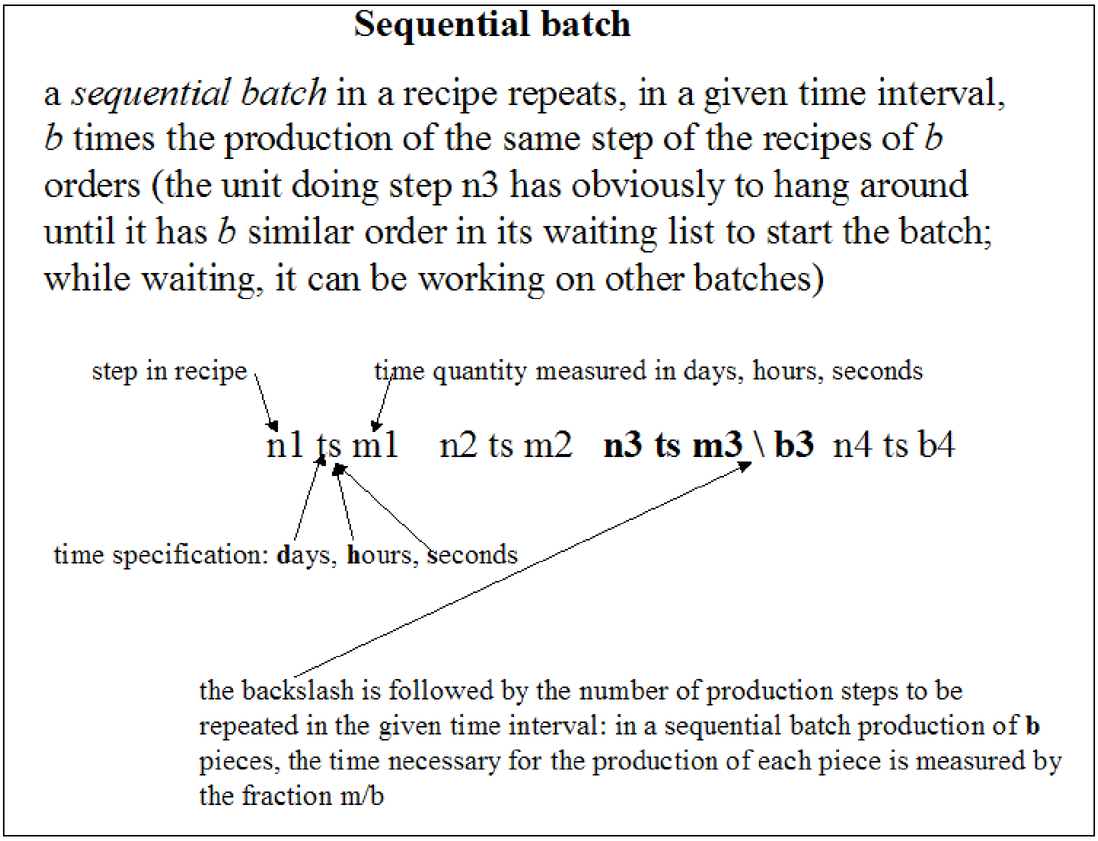
[2] In the site we can read that: “The NIIIP Consortium consists of a group of leading United States information technology suppliers, industrial manufacturing end users, academic, and standards organizations with a common interest in developing an information infrastructure architecture to enable organizations 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.



Sequential batch

a *sequential batch* in a recipe repeats, in a given time interval, *b* times the production of the same step of the recipes of *b* orders (the unit doing step n3 has obviously to hang around until it has *b* similar order in its waiting list to start the batch; while waiting, it can be working on other batches)

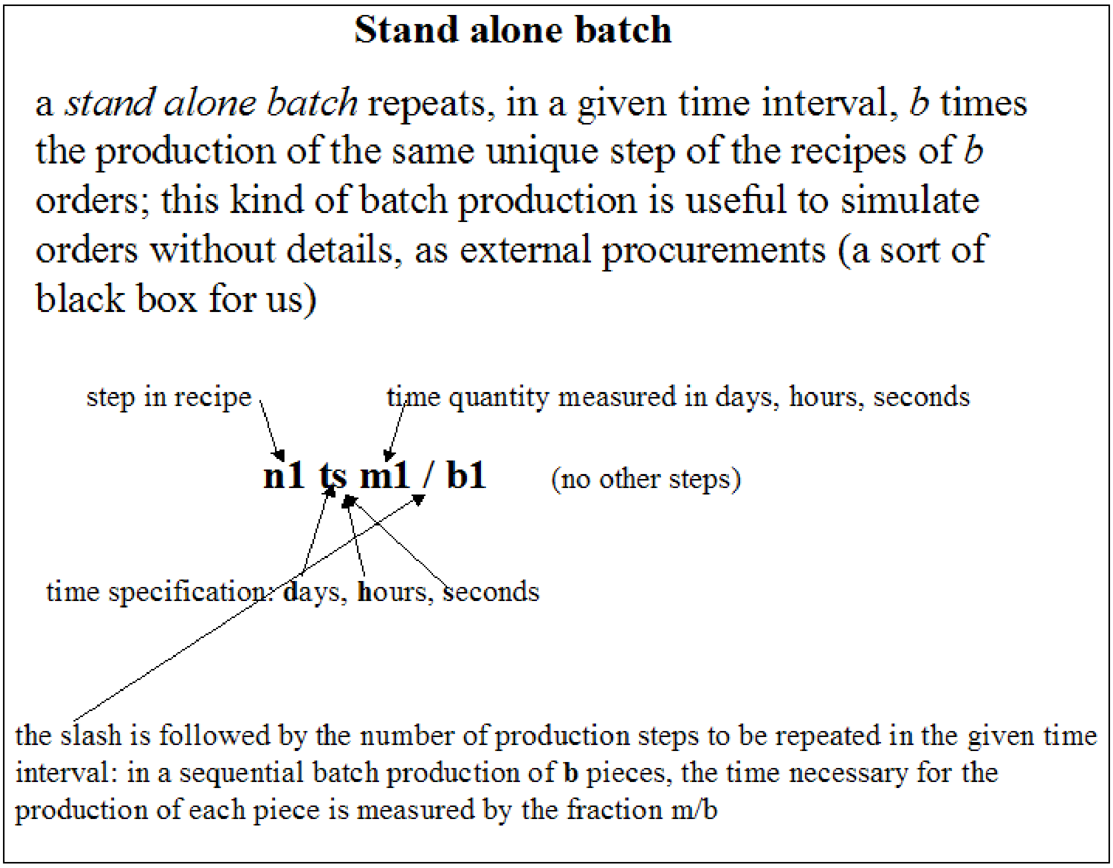
step in recipe
time quantity measured in days, hours, seconds
n1 ts m1 n2 ts m2 **n3 ts m3** \ **b3** n4 ts b4
time specification: **d**ays, **h**ours, **s**econds

the backslash is followed by the number of production steps to be repeated in the given time interval: in a sequential batch production of **b** pieces, the time necessary for the production of each piece is measured by the fraction m/b

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.



Stand alone batch

a *stand alone batch* repeats, in a given time interval, *b* times the production of the same unique step of the recipes of *b* orders; this kind of batch production is useful to simulate orders without details, as external procurements (a sort of black box for us)

step in recipe
time quantity measured in days, hours, seconds
n1 ts m1 / b1
(no other steps)
time specification: days, hours, seconds

the slash is followed by the number of production steps to be repeated in the given time interval: in a sequential batch production of **b** pieces, the time necessary for the production of each piece is measured by the fraction m/b

Figure 6. Stand alone batch.

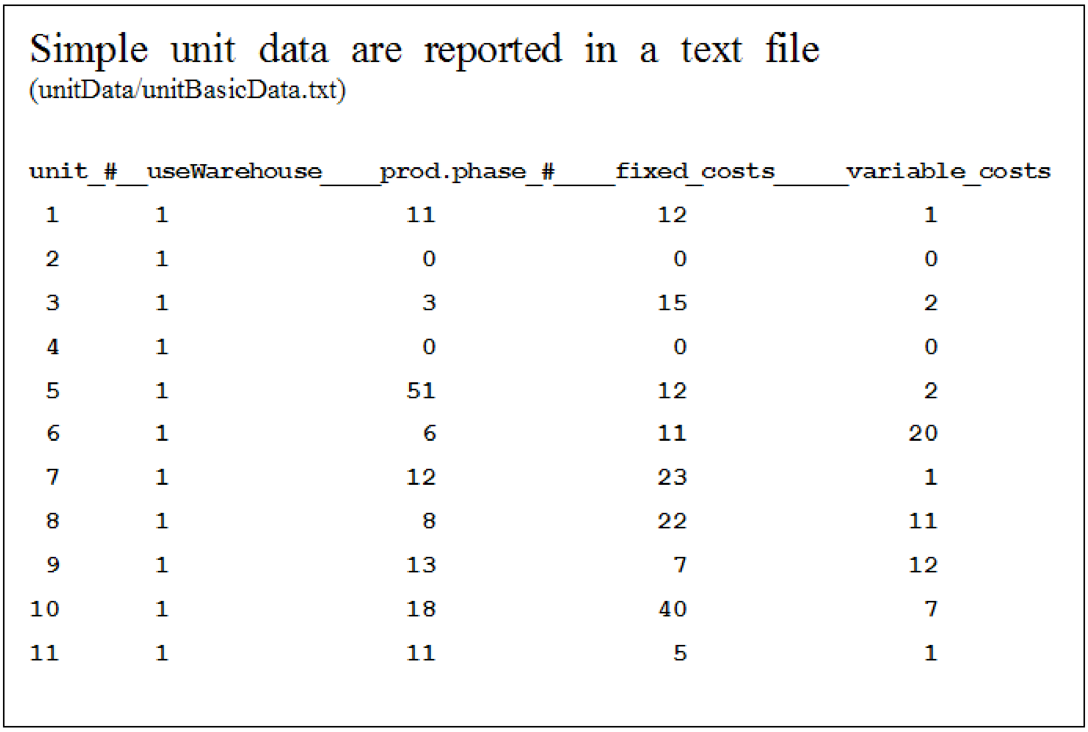
A closer look to the DW side

@@@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)

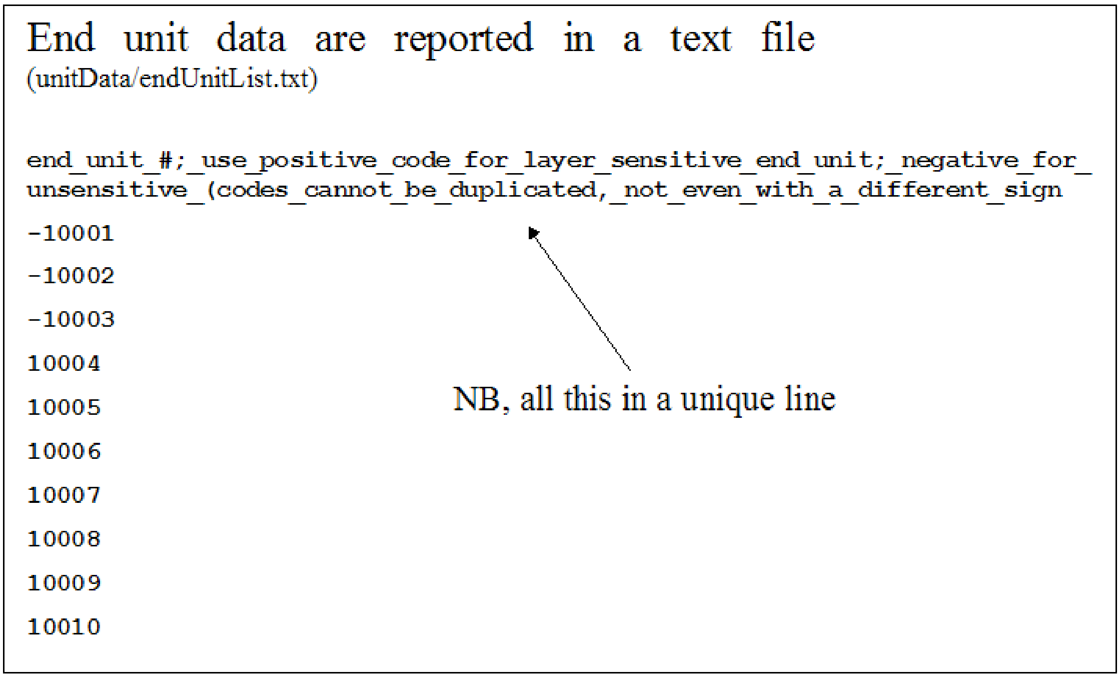
```
unit_#__useWarehouse____prod.phase_#____fixed_costs_____variable_costs
 1        1               11          12           1
 2        1                0           0           0
 3        1                3          15           2
 4        1                0           0           0
 5        1               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



End unit data are reported in a text file
(unitData/endUnitList.txt)

```
end_unit_#;_use_positive_code_for_layer_sensitive_end_unit;_negative_for_
unsensitive_(codes_cannot_be_duplicated,_not_even_with_a_different_sign
-10001
-10002
-10003
10004
10005
10006
10007
10008
10009
10010
```

NB, all this in a unique line

Figure 9. End units ???.

To be implemented: units locking resources for other units

Warehouses

Newses

Newly back to the WD side

Procurements, OR, AND; layers, memory matrixes

Computational capabilities and memory matrixes

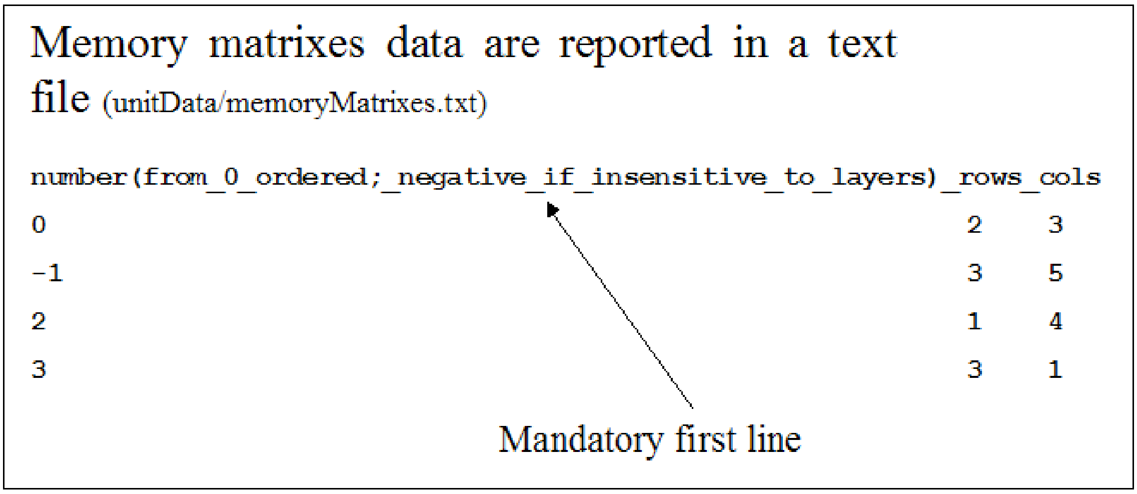
jVE has computational capabilities that can be associated to each step of a recipe. To use this feature of the program it is necessary to understand Java language, as we have to modify[3] the ComputationalAssembler.java file. The goal of the computational capabilities is that of dealing with forecasting,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

---

[3] We have not to modify the original file, which is included in the src/ folder in the main folder (./) of the program; copy instead the file from src/ to ./ and modify the copy; the 'make run' command uses the classes contained in lib/jveframe.jar (which are those contained in src/), but the classes in ./ overriding those in jveframe.jar.

Computational results are also used as a choice criterion in OR sequences (not yet implemented).

Computations use data contained in memory matrixes create following both the totalMemoryMatrixNumber of the model probe (this parameter can be also set via the jveframe.scm file) and the contents of the file unitData/memoryMatrixes.txt shown in Figure xa. Memory matrixes uses 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. Here the second matrix (numbered 1, being 0 the number of the first one) in insensitive to layers



Memory matrixes data are reported in a text file (unitData/memoryMatrixes.txt)

```
number(from_0_ordered;_negative_if_insensitive_to_layers)_rows_cols
0                                                          2   3
-1                                                         3   5
2                                                          1   4
3                                                          3   1
```

Mandatory first line

Figure xa. Memory matrixes declarations.

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

As seen above (*not jet 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.

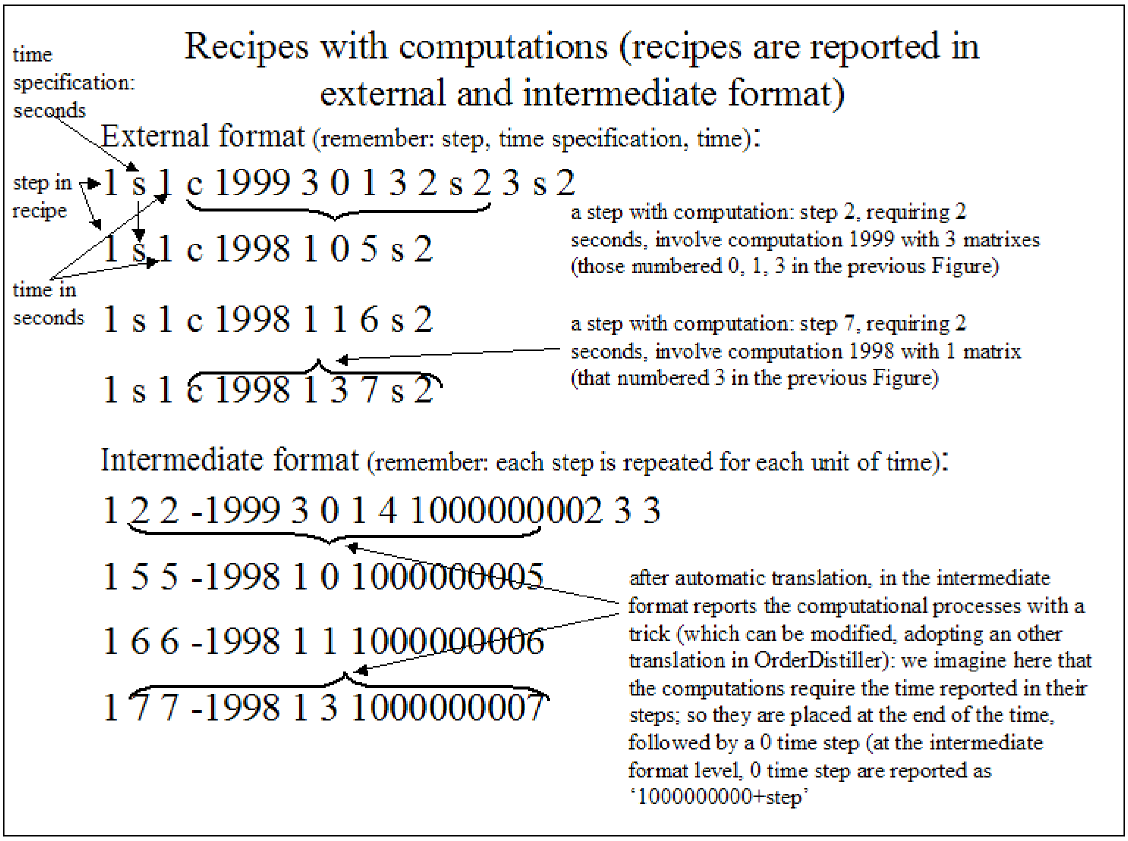
The format of a computation is: ‘c *code n* m_1 … m_n’ where ‘c’ is mandatory, ‘*code*’ is the code of the computation, ‘*n*’ is the number of matrixes to be used and ‘m_1 … m_n’ are the numbers of those matrixes, as reported in the file unitData/memoryMatrixes.txt (Figure xa).

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 you have to add lines similar to those introduced below (Figure xc1 and xc2) into the ComputationalAssembler class.

In Figure xb we can see how computational codes are represented following their format. 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), at 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), at the proper layer, as above. In the Java code of Figure xc1 we can see those operation made upon a symbolic mm0 (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 'done'.


Recipes with computations (recipes are reported in external and intermediate format)

External format (remember: step, time specification, time):

1 s 1 c 1999 3 0 1 3 2 s 2 3 s 2

1 s 1 c 1998 1 0 5 s 2

1 s 1 c 1998 1 1 6 s 2

1 s 1 c 1998 1 3 7 s 2

a step with computation: step 2, requiring 2 seconds, involve computation 1999 with 3 matrixes (those numbered 0, 1, 3 in the previous Figure)

a step with computation: step 7, requiring 2 seconds, involve computation 1998 with 1 matrix (that numbered 3 in the previous Figure)

Intermediate format (remember: each step is repeated for each unit of time):

1 2 2 -1999 3 0 1 4 1000000002 3 3

1 5 5 -1998 1 0 1000000005

1 6 6 -1998 1 1 1000000006

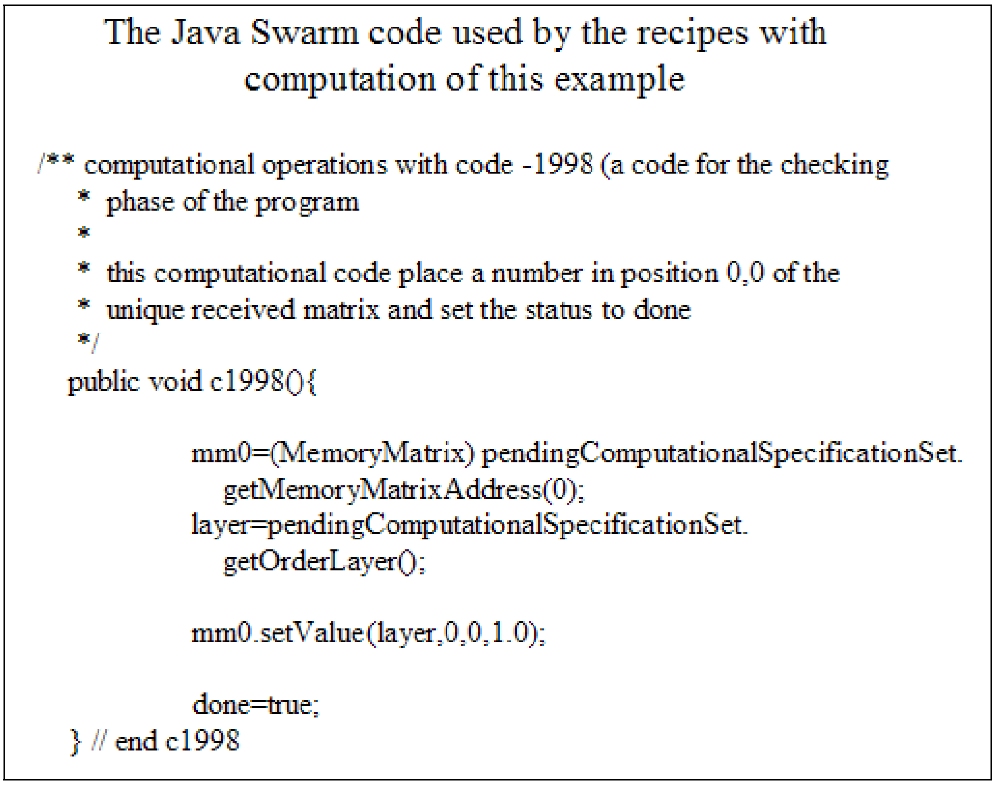
1 7 7 -1998 1 3 1000000007

after automatic translation, in the intermediate format reports the computational processes with a trick (which can be modified, adopting an other translation in OrderDistiller): we imagine here that the computations require the time reported in their steps; so they are placed at the end of the time, followed by a 0 time step (at the intermediate format level, 0 time step are reported as '1000000000+step'

Figure xb. The format of the computational processes.

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 'done' the computational step. Into this code 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’ (executed at least at step 5 or 6 or 7 deepness).

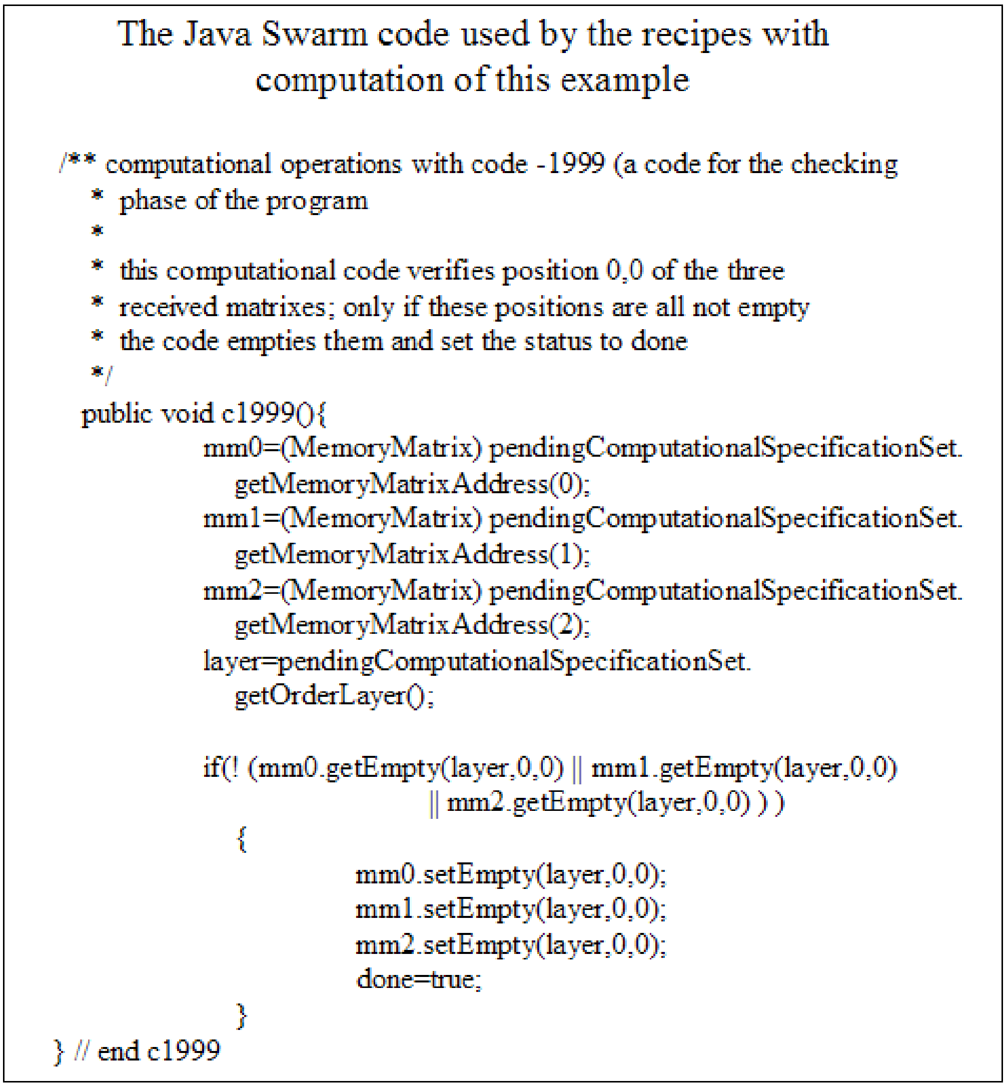


The Java Swarm code used by the recipes with
computation of this example

```
/** 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.

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



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.

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 testNB spiegare l'uso del distillere di B&B&B rispondendo a "ma come si passano le ricette"

NB How to get jVE