FP6-034691

Net-WMS

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Networked Businesses

D5.2 – Specification of the Collaborative System between User and Optimisation module

Due date of deliverable: 15 April 2008

Actual submission date: 15 April 2008

Start date of project: 1 September 2006 Duration: 36 months

Organisation name of lead contractor for this deliverable: INRIA Version 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level : PU

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	**Net-WMS**
Project Full Name:	Towards integrating Virtual Reality and optimisation techniques in a new generation of **Net**worked businesses in **W**arehouse **M**anagement **S**ystems under constraints
Document id:	D5.2
Document name:	Specification of the Collaborative System between User and Optimization Module
Document type (PU, INT, RE, CO)	PU
Version:	1
Submission date:	15-04-2008
Authors: **Organisation:** **Email:**	F. Fages & J. Martin INRIA Francois.Fages@inria.fr

Document type PU = public, INT = internal, RE = restricted, CO = confidential

ABSTRACT:

This deliverable specifies the collaborative system between the user and the packing optimization solver. After an introduction, and a presentation of the optimizer component in the Net-WMS architecture, we describe the user interaction capabilities with the packing optimization solver, first with an example of packing problem solving taken from the PSA study cases, and then through a complete specification of the communication between the optimizer and the virtual warehouse. All these communications are expressed in the Packing Knowledge Modelling Language PKML defined in the project, and will be implemented in PackXML, the markup language used for exchanging PKML expressions between components.

KEYWORD LIST: User Interaction, Optimization, Constraint Solving, Visualization.

MODIFICATION CONTROL			
Version	Date	Status	Author
0	31-01-2008	Initial	F. Fages
1	12-04-2008	Complete	F. Fages
2	15-04-2008	Final	F. Fages

Deliverable manager

- François Fages, INRIA

List of Contributors

- François Fages, INRIA
- Julien Martin, INRIA
- Abder Aggoun, KLS-OPTIM
- Camille Chigot, CEA
- Philippe Gravez, CEA

List of Evaluators

- Camille Chigot, CEA
- Alessandro Leverano, CRF

TABLE OF CONTENT

1 Introduction

Net-WMS aims at optimizing the packing processes (planning, packing and unpacking) in warehouses so as to save shipping space and perform faster and better. In packing activities, several levels of packing must be considered from the boxes that are typically provided by suppliers to the containers that are actually shipped as single units. In the automotive industry, storing a simple part in a container involves two or three packing levels as described in Net-WMS deliverable D2.2 "Case Studies". This multi-level packing process is difficult to manage and Net-WMS intents to demonstrate the benefits that optimization and virtual reality (VR) techniques may bring to Warehouse Management Systems (WMS). These two techniques are fully complementary: the former mainly focuses on automated decision-making assistance while the latter emphasizes interactivity with the human experts in charge of planning the packing processes.

Net-WMS Work Package WP5 "Virtual Reality applied to packing problems in a WMS" addresses three main issues:

1. the 3D visualisation of the packed and packing objects allowing human operators to handle them "virtually"
2. the interactive (i.e. real-time with respect to human perception) simulation of the physics laws ruling these objects in order to "virtually rehearse" packing and unpacking operations, as well as to dynamically study the behaviour of container contents during shipping
3. mixing the capabilities of optimisation and virtual reality to provide the most efficient assistance to the human users. These capabilities will help, on the one hand, to design new models and validate the correctness and sufficient completeness of packing business rules, and on the other hand, to find solutions in difficult cases.

For WP5, the main objective of this second year is to incrementally build the main Virtual Warehouse module which will be the user interface to all the optimisation, virtual reality and working knowledge capabilities. For this objective, the first step is the definition of the specification of the three above points. This is divided in two deliverables:

- **D5.3**: “Specification of the Virtual Warehouse API” (M18) specifies the points over visualisation and interactive simulation (1st and 2nd points).
- **D5.2**: “Specification of the collaborative system between user and optimisation module” (M18) specifies the integration between the Warehouse API and the solver (3rd point)

The current deliverable document D5.2 "Specification of the Collaborative System between User and Optimization Module" describes the interaction capabilities of the optimizer component with the different users, and the interface of this component with the virtual warehouse. The specifications in this document are closely related to the Packing Knowledge Modelling Language (PKML) [Deliverable D6.1], since PKML provides a pivot language for exchanging packing knowledge and data between Net-WMS components, and thus in particular between the optimizer and the virtual warehouse with which the users interact.

The goal of this document is twofold:

1. on the one hand, to illustrate the user interaction capabilities with the packing optimizer component developed in Net-WMS;
2. on the other hand, to specify all the necessary communication between the virtual warehouse and the packing solver.

A mock–up has been implemented to serve these purposes by adapting the graphical user interface for constraint logic programming CLPGUI [Fages Soliman Coolen 04] to the PKML modelling language and to dedicated 3D packing visualization. It has been presented several times to the end users of the project to focus discussion on concrete examples and facilitate the communication of the industrial requirements. This mock-up constitutes a proof-of-concept implementation of the prototype of the collaborative system that will be developed next year on the basis of [Deliverable D5.3] and delivered in Deliverable D5.5 at M28.

We are in the middle of Task T5.3 which still involves meetings and discussions with the end-users (WP2), the language designers (WP6) and the constraint solver makers (WP4). It is however already clear that the interaction capabilities of the optimisation module described in this report will provide significant advances in optimizing strategies that go well beyond what is currently possible in warehouse management software systems [Deliverable D3.1].

2 The Packing Optimizer Component in the Net-WMS Architecture and its Different Users

2.1 Net-WMS Architecture

Let us recall in Figure 1 the general architecture proposed in Net-WMS for packing components. In this deliverable we will highlight the connection between the packing optimization solver used for business applications, and the presentation layer used by the different end-users of the system.

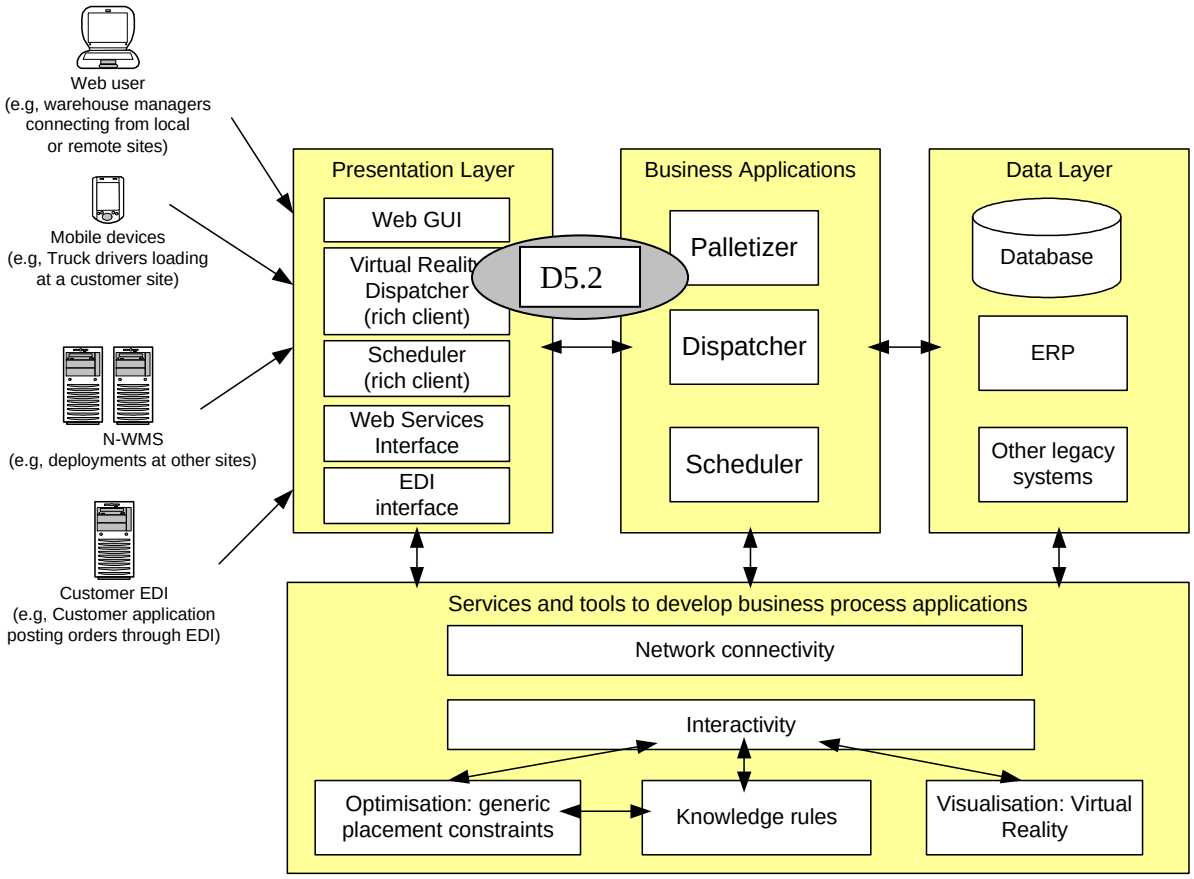


Figure 1: **General Net-WMS Architecture**

The Virtual Warehouse described in Deliverable D5.3 is the main constituent of the presentation layer. It is the module that makes the link between the users, the virtual reality, the optimizers and the packing problems. Three categories of users will interact with the system:

1. The Warehouse expert is a person having a deep experience in packing. He is able to structure and describe his knowledge in term of business rules. He is responsible for the definition of the knowledge rules that will be used by further users.

2. The Warehouse manager is a person able to design packing problems according to the orders of the client of the warehouse by exploiting the knowledge rules written for the optimization solver. He has the capability to change and to adapt, if required, the packing results using optimization and virtual warehouses tools.
3. The Warehouse planner is the main end-user of the virtual warehouse. He will have to solve packing problems in order to prepare packing orders for operational staff.

Their respective interaction modes are depicted in Figure 2 below. All Man Machine interactions are transcribed into PKML files or statements (using appropriate editors and database imports and exports), while all interaction between software components will be done in PackXML, the markup language for exchanging PKML expressions between components over the network.

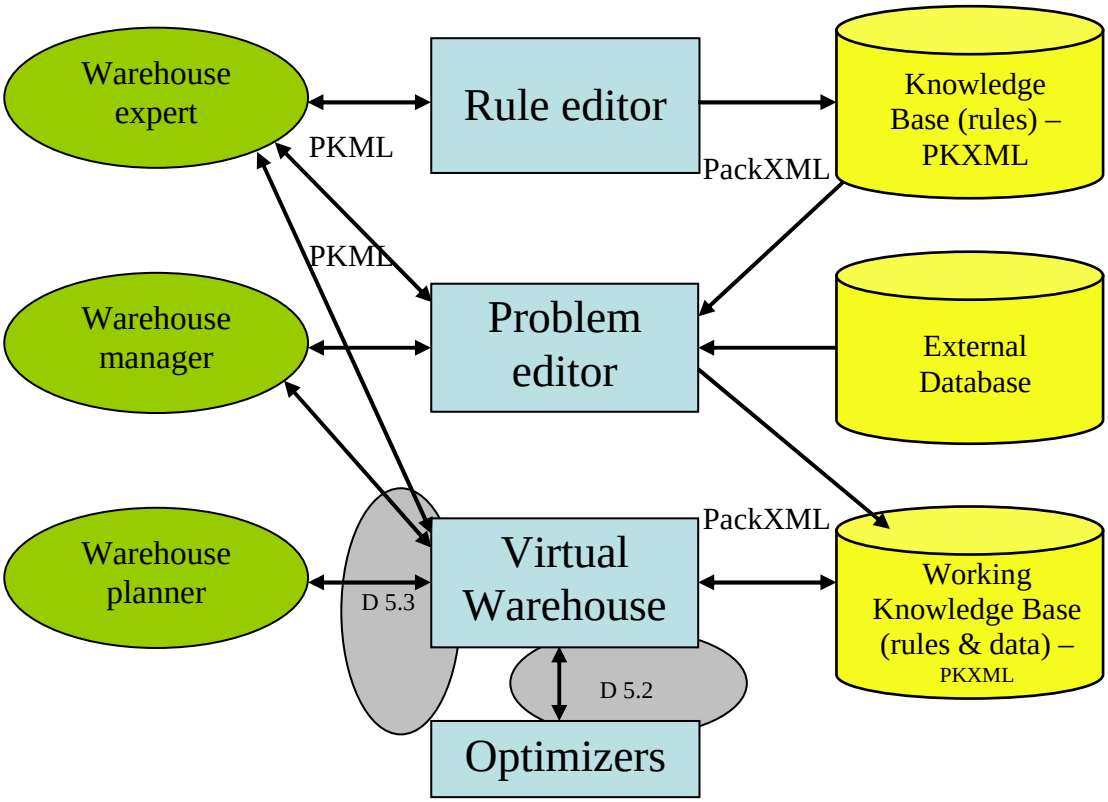


Figure 2 The Virtual Warehouse with its three categories of users.

2.2 Packing Solver

As explained in Deliverable D6.1, the business packing component is structured into subcomponents depicted in Figure 3:

- Packing logic: this is the main entry to drive the different components of the packing module.
- Packing solver: the optimizer; this components takes well defined inputs and produces results which are then processed by the packing container.
- Packing player: this is 3D visualisation of containers and items of the container (from which 2D projections can be printed on a paper sheet for exploitation).

- Packing designer: this module allows the user to design packing modules. The user can edit a configuration, set items and launch the solver to complete packing.

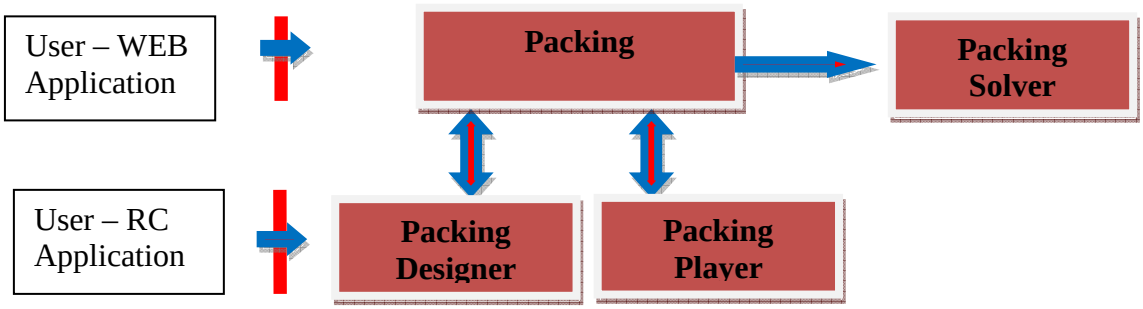


Figure 3: Architecture of the packing component

This deliverable focuses on the *packing design* module and specifies the required communications between the *packing solver* and the *(3D) packing player* in order to implement the collaborative system between the users and the packing optimizer. Again, the Packing Knowledge Modelling Language (PKML) is used to specify packing rules and data, and PackXML is used to exchange packing knowledge between all Net-WMS components. PKML will thus be used with appropriate editors or database imports by the Warehouse Expert and the Warehouse Manager to enter packing business rules and packing application data into the virtual warehouse. The XML exchange format for this language PackXML will be used for all communications between components and with databases. In particular, PackXML will thus be used for all communication between the virtual warehouse and the optimisation packing solver referred here as PKML solver.

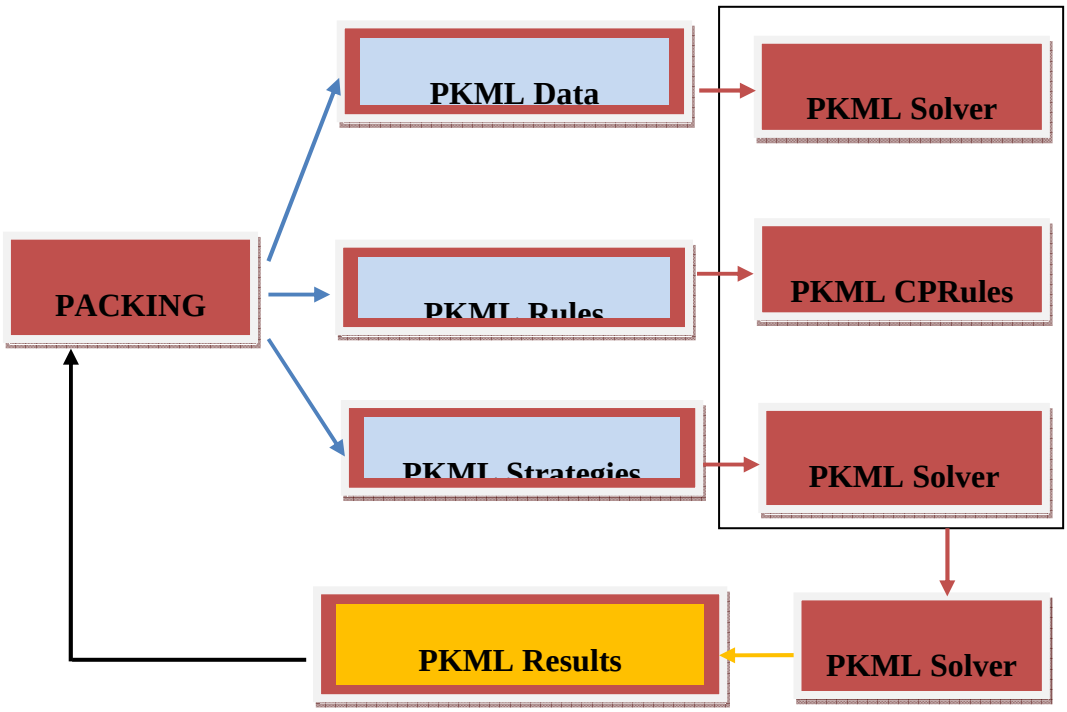


Figure 4: PKML statements used in the packing component

The Net-WMS project expects to provide two implementations of the PKML solver. The first implementation uses the PKML compiler toward SICTUS Prolog [Carlsson et al. 07] for rapid prototyping in Constraint Logic Programming, and the second one will use the PKML compiler toward the open source Choco solver in Java, for better integration, flexibility and extensibility according to application needs.

3 Specification on the Collaborative System with a Running Example

3.1 Presentation of the Example and of the Mock-up

This section describes a complete example showing, successively, the packing knowledge model in PKML (including strategies, patterns, constraints, etc.), the packing data in PKML (including shapes, items, bins, etc.), the packing problem solving (placements), and the various visualizations and interactions with the packing solver. These constitute the main plug-in components of the Packing Solver recalled in Figure 5.

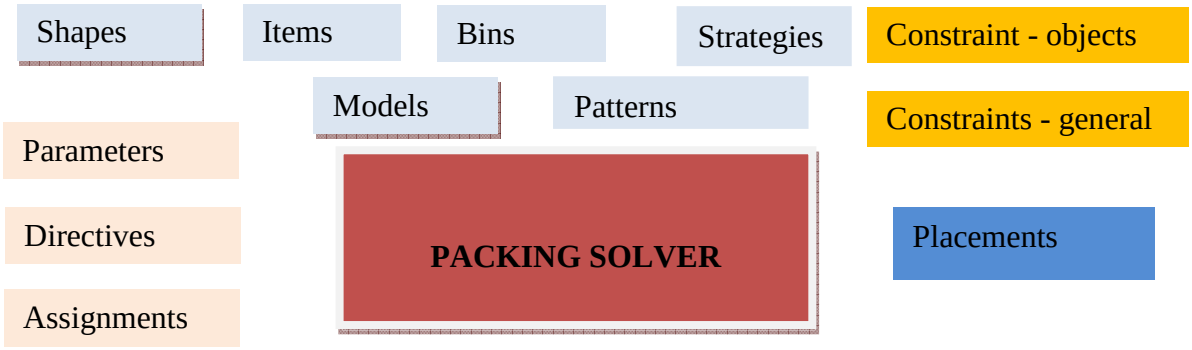


Figure 5: General plug-in components of the Packing Solver

The running example of this report is a container loading problem inspired from the PSA study case for pallet loading [Deliverable D2.2 section 2]. The problem is to find a placement of cartons in a container such that, in addition to the non-overlapping constraints of a pure bin packing problem, the following further common sense and industrial requirements are satisfied by the packing:

- Gravity rules (any item must be on top of another one or on the ground of the container),
- Stacking rules w.r.t. weights (an item on top of another one must be lighter),
- Weight balancing rules (the weight supported by the two halves along the length of the container must not exceed a desired ratio),
- Stacking rules w.r.t. sizes (the difference of size between stacked items must be less than a given value) ,
- Packing business patterns embedding other stability requirements.

For the sake of illustration, the mock-up uses the generic graphical user interface for constraint programming CLPGUI [Fages Soliman Coolen 04] connected to a PKML solver compiled [Deliverable D6.1 section 4.3] in SICStus Prolog [Carlsson et al. 07]. Figure 6 depicts the command console of CLPGUI which provides the functionalities of this architecture for:

- Connecting to the solver,
- Executing PKML goals in the command line
- Executing predefined PKML goals by clicking on buttons
- Backtracking for computing other solutions
- Backtracking in the history of interactions
- Clearing the current PKML definitions sent to the packing solver.

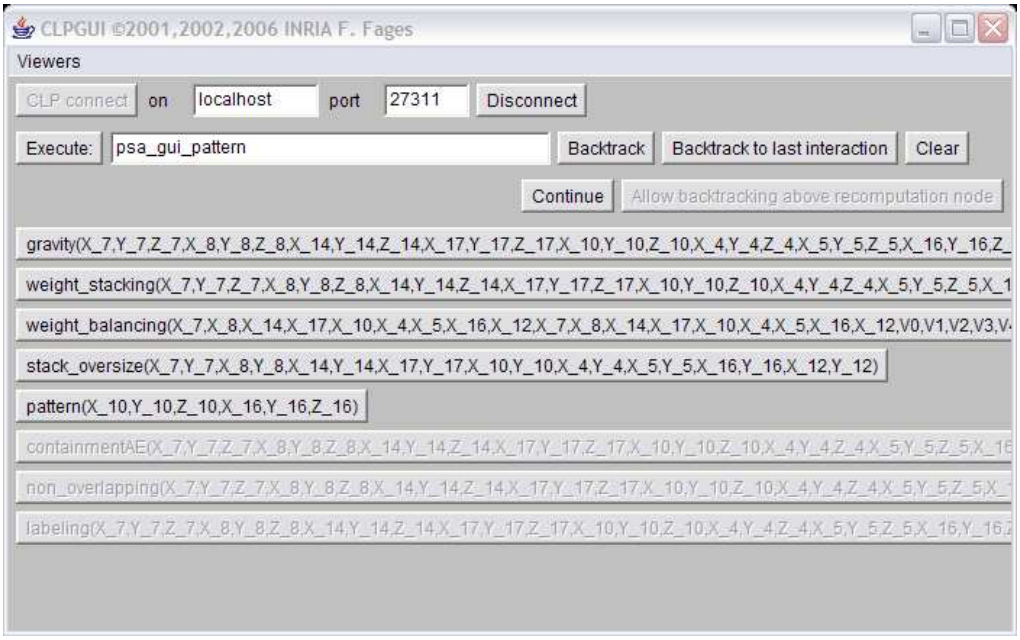


Figure 6: Main console of the CLPGUI interface with the PKML Solver

A solution computed for this problem is depicted in Figure 7.

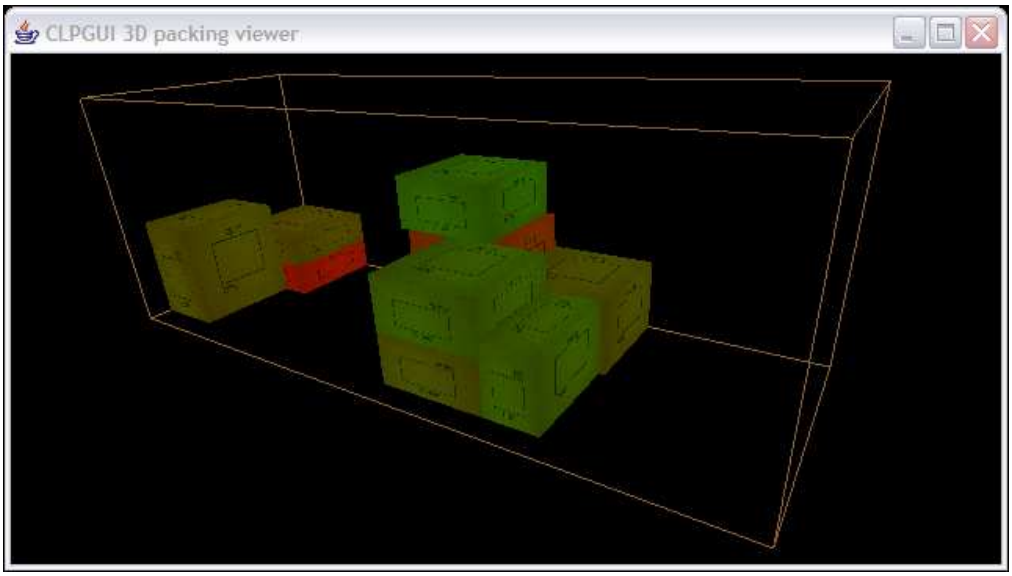


Figure 7: 3D Visualization of a solution in CLPGUI

3.2 Uploading a Packing Knowledge Model

The packing knowledge models including the packing requirements and the nomenclatures of shapes are typically specified by the Warehouse Expert, and adapted to new situations by the Warehouse Manager. Here the knowledge model used for the example is specified by the following Rules2CP/PKML file named psa_knowledge.rcp (note that appropriate editors of Rules2CP/PKML files are assumed but not discussed in this report):

```
import(pkml).

solve(Items, Bin, Patterns, Dims) -->
        gravity(Items) and
        weight_stacking(Items) and
        weight_balancing(Items, Bin, 1, 20) and
        stack_oversize(Items, 20) and
        pattern(Items, Patterns Dims) and
        bin_packing(Items, [Bin], Dims).

s1 = {size=[1400, 500, 500]}.
s2 = {size=[224, 224, 222]}.
s3 = {size=[224, 224, 148]}.
s4 = {size=[224, 224, 111]}.
s5 = {size=[224, 224, 74]}.
s6 = {size=[155, 224, 222]}.
s7 = {size=[112, 224, 148]}.

pat1 = {sids=[s2, s3], origins=[[600,0,0], [824,0,0]]}.
```

patterns = [pat1].

The rules for gravity, weight_stacking, weight_balancing, stack_oversize, as well as the rule for business packing patterns, are predefined in the PKML library, see [Deliverable D6.1 section 4.4.3]. In addition, a nomenclature of 7 shapes and one (toy) pattern for placing objects of shapes s2 and s3 are defined in this file.

Moreover, for the sake of the connection to CLPGUI, an additional Rules2CP file named gui.rcp defines several possible command buttons for the main console. The status in PKML of these commands for the GUI is the status of builtins of the target language. They are treated as such by the compiler with simple inline rules.

```
psa_gui_pattern2 -->
        gui_control(button, solve(items, [bin], patterns, dims))
        and gui_names(items, [bin], dims).

psa_gui_pattern -->
        gui_control(button, gravity(items)) and
        gui_control(button, weight_stacking(items)) and
        gui_control(button, weight_balancing(items, bin, 1, 20))
         and gui_control(button, stack_oversize(items, 20)) and
        gui_control(button, pattern(items, patterns, dims)) and
        gui_names(items, [bin], dims) and
        bin_packing_gui(items, [bin], dims).

bin_packing_gui(items, bins, dims) -->
        gui_control(button, containmentAE(items, bins, dims)) and
        gui_control(button, non_overlapping(items, dims)) and
        gui_control(button, labeling(items)).
```

The execution of the first rule creates a console with only one button solve as in Figure 8. The second rule creates the console depicted in Figure 6 with buttons for the packing business rules taking into account weights and size constraints. The last rule defines buttons for the pure bin packing rules [Deliverable D6.1 section 4.4.2].

3.3 Uploading a Packing Problem Dataset

For dealing with a particular packing problem instance, the items, bins and the packing problem type are edited, or imported from an external database, into a dataset file that is uploaded to the PKML solver. This is the routine job of the Warehouse Planner. The dataset file used for our running example is the following Rules2CP file data1.rcp :

```
import(psa_knowledge).
import(gui).
```

```
i1 = {sid=s1, origin=[0,0,0]}.

i2 = {sid=s4, origin=[_,_,_], weight=413}.
i3 = {sid=s5, origin=[_,_,_], weight=463}.
i4 = {sid=s5, origin=[_,_,_], weight=842}.
i5 = {sid=s3, origin=[_,_,_], weight=422}.
i6 = {sid=s4, origin=[_,_,_], weight=266}.
i7 = {sid=s4, origin=[_,_,_], weight=321}.
i8 = {sid=s2, origin=[_,_,_], weight=670}.
i9 = {sid=s6, origin=[_,_,_], weight=440}.
i10 = {sid=s7, origin=[_,_,_], weight=325}.

bin = i1.
items = [i4,i8,i3,i9,i5,i2,i10,i7,i6].
dimensions = [1,2,3].
```

This file specifies 10 objects, 1 bin and 9 items to pack in one bin. This constitutes the running example of this section. The execution of the PKML goal

```
? psa_gui_pattern2.
```

in the CLPGUI console creates only one button solve, as depicted in Figure 8.

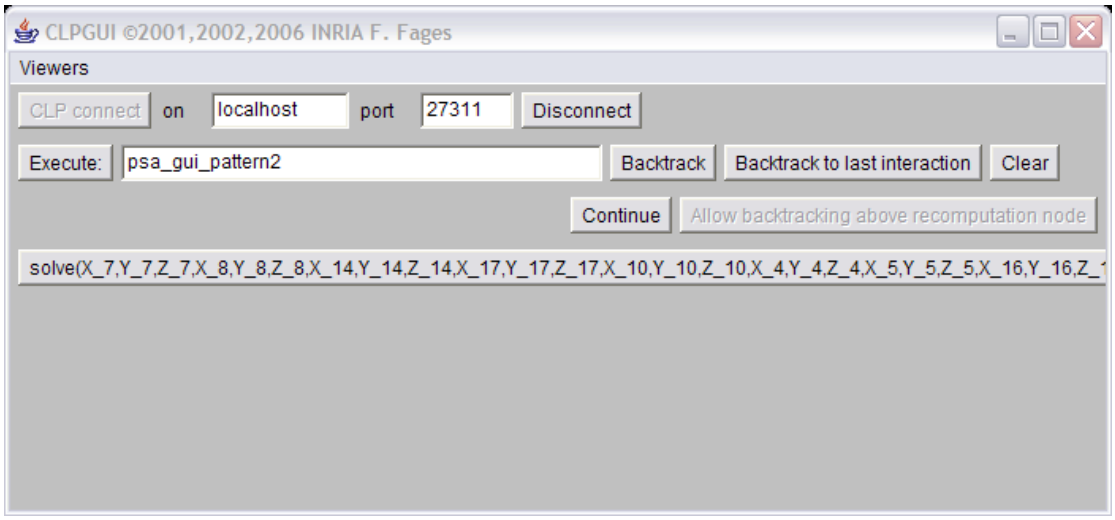


Figure 8: Main CLPGUI console created by the goal psa_gui_pattern2

3.4 Solving Request

In this first scenario, all packing rules are posted by the solve rule. The user requests the computation of a solution by clicking on the solve button of the CLPGUI console (Figure 8), and can visualize the computed solution in the CLPGUI 3D packing viewer (Figure 7). Such a solution can then be saved in the virtual warehouse in a database, printed in an appropriate

form for the exploitation (e.g. a couple of 2D projection maps on a paper sheet at PSA). In this communication protocol, all directives from the GUI to the solver for requests, and vice-versa for answers, are PKML statements exchanged between components possibly over the network in PackXML format.

This simple use of the system for getting a packing solution automatically is the routine job of the Warehouse planner. The functions in the following may help him further to improve the solution.

3.5 Request for other Solutions

Other solutions, such as the solution depicted in Figure 9 below, can be obtained by clicking on the “Backtrack” button of the CLPGUI console.

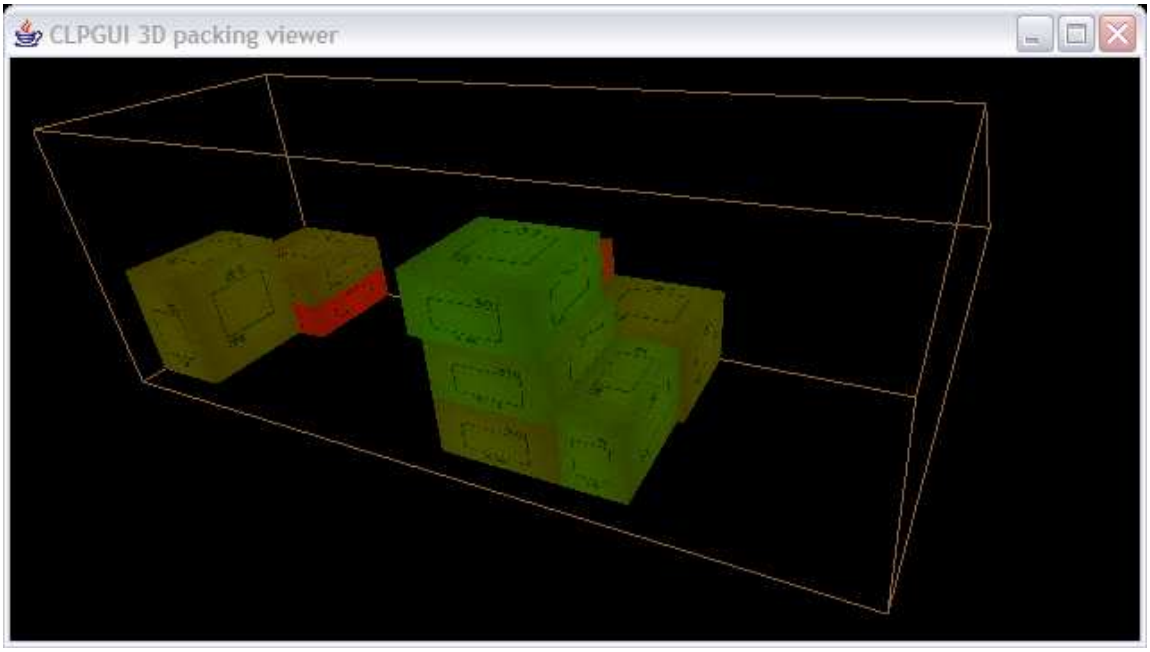


Figure 9: Other solution obtained for the same problem.

3.6 Modifying a Configuration

The 3D packing viewer of solutions should not be only a visualization component but an interactive component in which the user can modify a configuration manually, by selecting objects and moving them. This is important to improve a given placement according to criteria not expressed in the PKML model, or for obtaining solutions not found by the packing solver.

Such manual modifications should be possible even if they violate packing rules and lead to configurations that are not solutions of the PKML packing model. These configurations are

represented in PKML with the same data structure as solutions [Deliverable D6.1] and are communicated in PackXML format.

3.7 Request for Getting Close Solution

After modifying the current packing of the objects, the user may request the packing solver to obtain a solution close to the manually obtained configuration. The solver checks whether it is a solution, in which case, it returns it as it is to the user, otherwise the solver returns a solution that minimizes the distance to the previous configuration. There are of course many ways to define such a distance between packing configurations and this is left to the warehouse expert in charge of designing the system. As an alternative, the current placement may also be used merely as a heuristics in order to avoid to deal with an minimization problem. Whatever the strategy is, this interaction involves a variant of the PKML rule for bin packing dealing with the notion of *current configuration* added to the packing knowledge model.

This feature is not implemented in the present mockup but was described for a similar placement problem in CLPGUI in [Fages Soliman Coolen 04].

3.8 Changing the Packing Rules

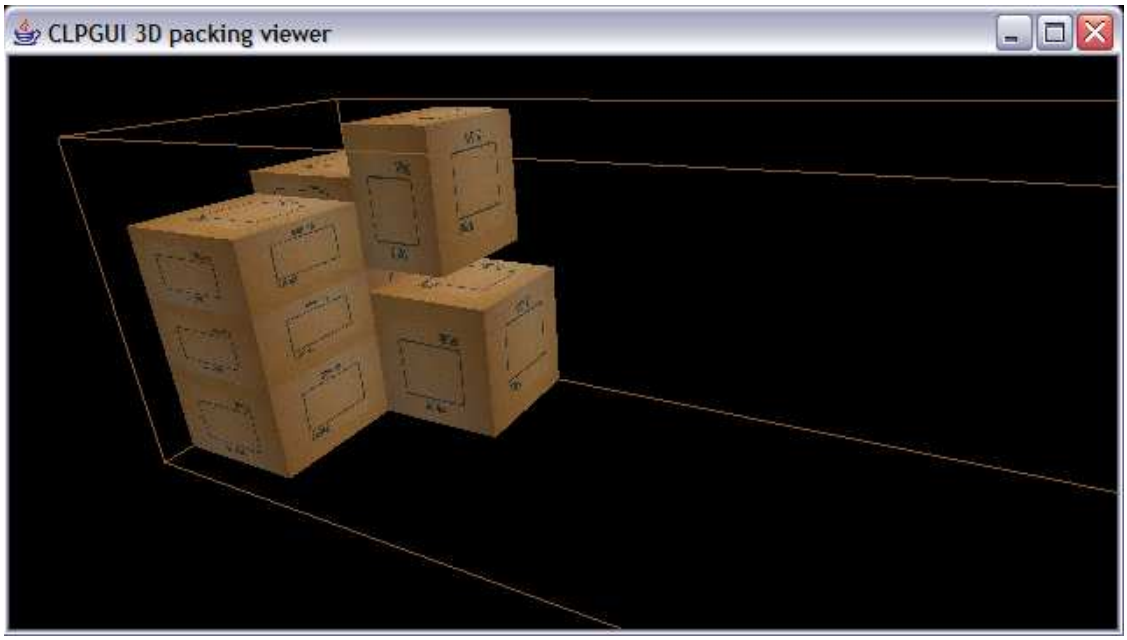
In the second scenario, the goal psa_gui_pattern creates a console with a button for each packing rule as depicted in Figure 6. This may be used to select or relax some packing rules and query the solver for computing solutions.

These functions may be used by the Warehouse planner in difficult cases where it is difficult to find a solution, and where some requirements may be violated. They are also instrumental for the Warehouse expert to help her design her packing knowledge models.

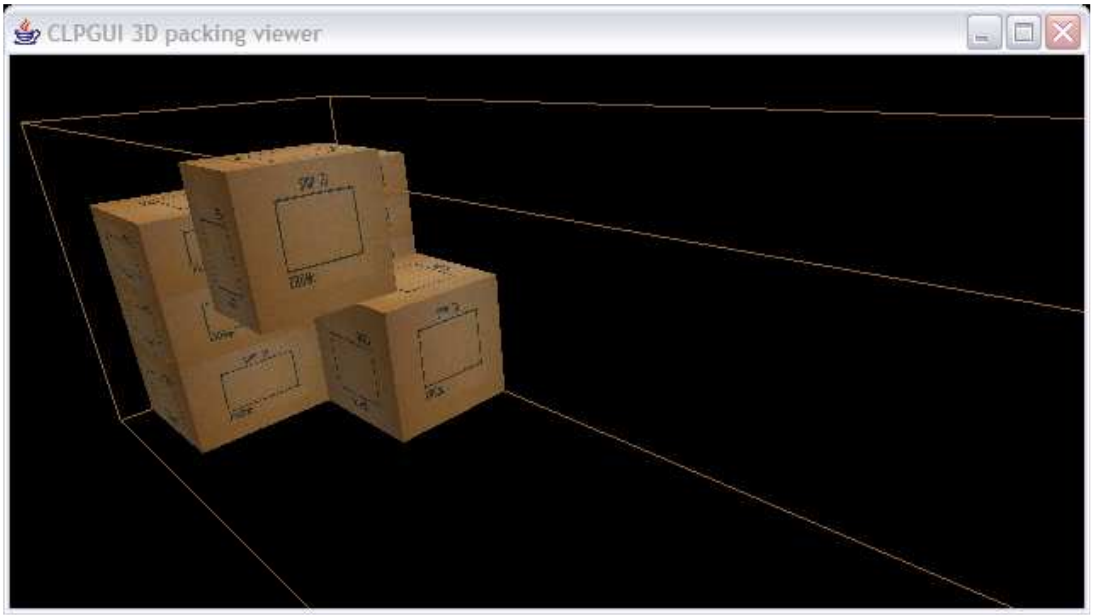
For instance, by first applying the set of rules corresponding to the pure bin-packing problem, namely by clicking on the buttons

- “containmentAE “ for enforcing boxes to be contained into the container.
- “non-overlapping” and
- “labelling” for searching for a solution

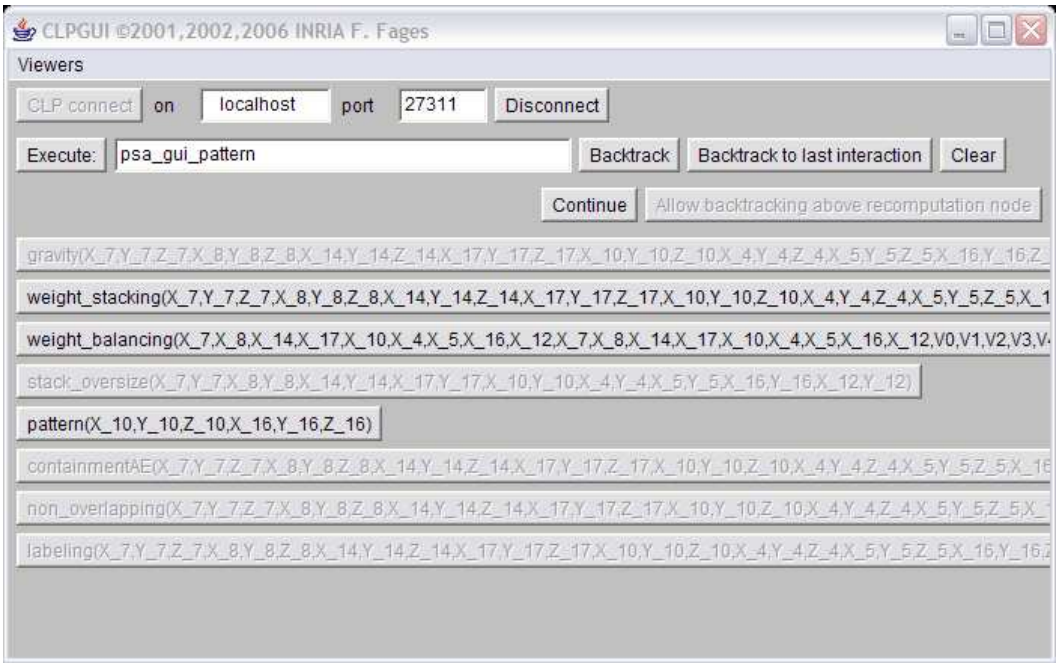
the packing solver can enumerate by backtracking solutions with floating cartons such as the following one:



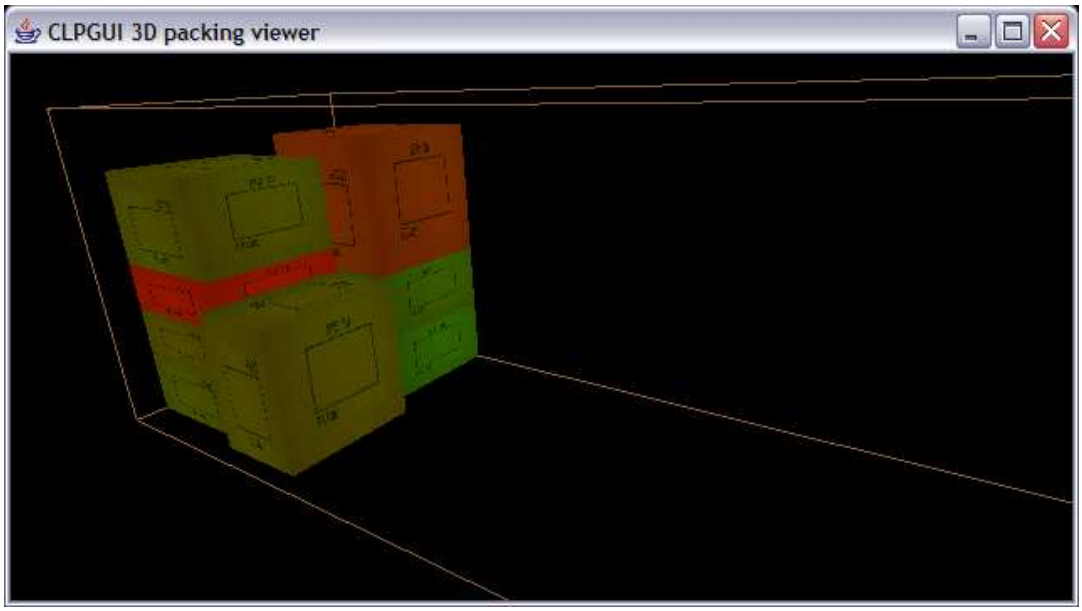
Adding the gravity rule with the console after backtracking to the labeling interaction, one can get non stable solution as:



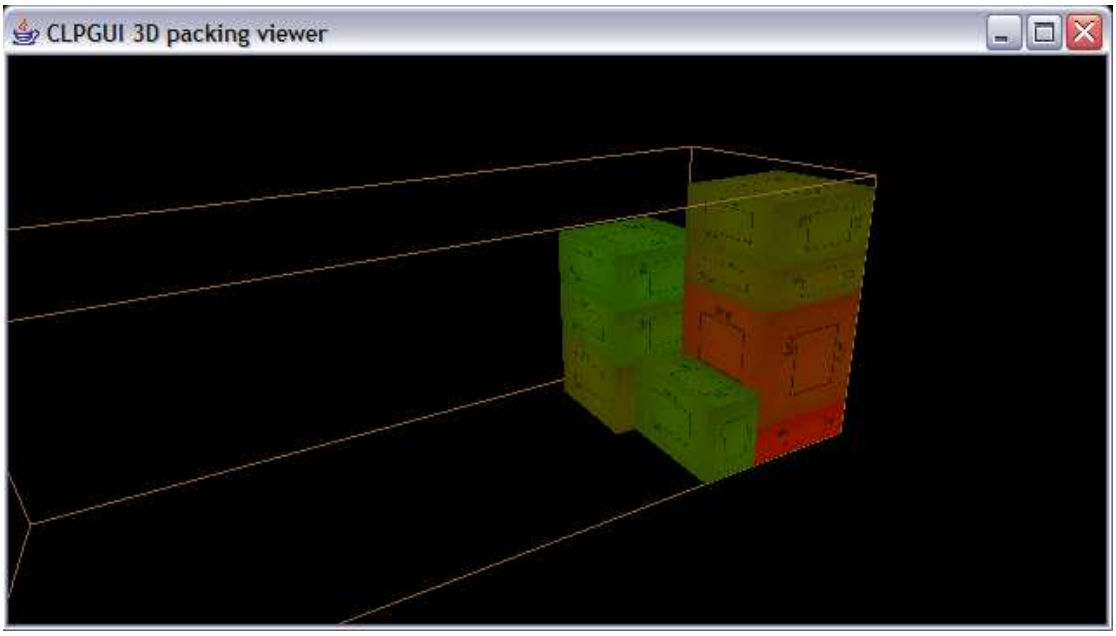
Now, adding the stack_oversize rule with the following console:



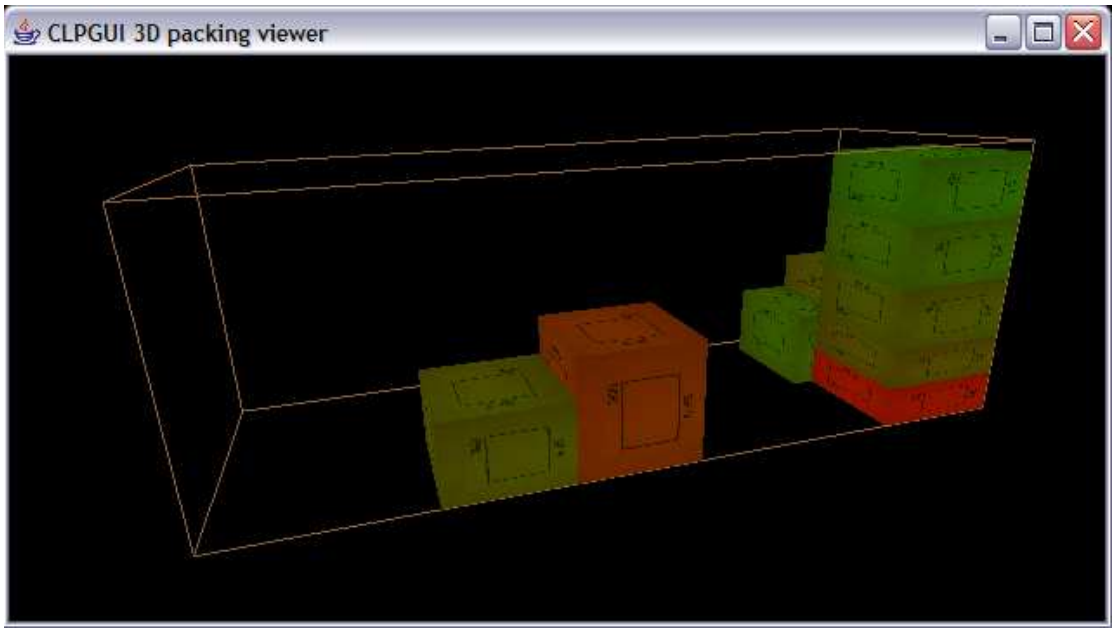
provides solutions like the following one:



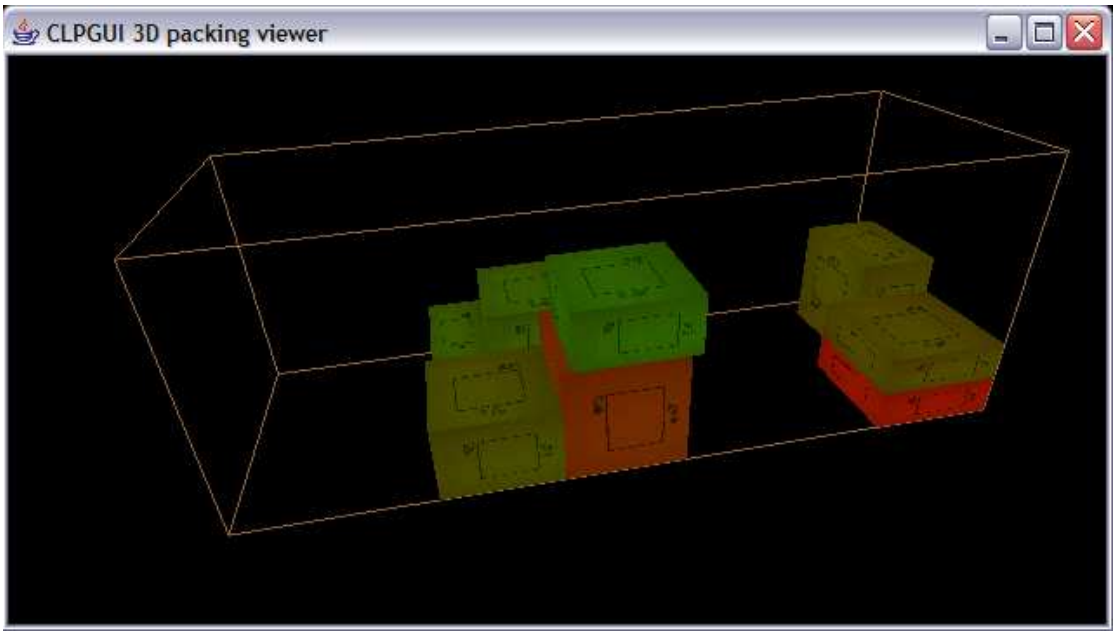
Such a solution may still violate the weight stacking rule (heavier items are depicted in red), which in turn can be enforced for getting solutions like the following one:



Furthermore, the pattern rule can be activated to place items of shape s2 and s3 in the middle of the container as specified in PKML by the toy pattern:



Now, the effect of the weight balancing rules is to equilibrate the container load with solutions like the following:



3.9 Other Requests

Since in this collaborative system, a request to the packing optimization solver is a PKML goal, virtually any request can be specified in the command line or in buttons of the GUI, provided the requested is defined in PKML with rules for dealing with it.

The results of the previous requests were complete solutions, but other information may be asked to the packing solver. For instance:

- The feasible/forbidden regions for an object,
- The placement of an object,
- Filling in a region,
- The creation of a hole in a region,
- Pushing the objects in one direction,
- Etc.

This information may be instrumental to the Warehouse Planner and to the Warehouse Expert to understand the difficulties of some problems and the effects of the requirements.

For the sake of simplicity of the implementation, the requests that would involve PKML statements as results, such as asking for the violated rules in the current configuration, are excluded from the current specification of the communication with the packing solver. It is worth noticing however that such PKML rules can be determined by trying to post them in order in the current configuration. This is what is recommended in the current specification for getting this kind of information.

4 Complete Specification

The collaborative system between the user and the optimization module is specified in this section by listing exhaustively the information types communicated from the virtual warehouse to the optimization component, and from the optimization component to the virtual warehouse.

In both cases, the transmitted information has the form of PKML files or statements, that will be exchanged in PackXML format, the markup language used for communicatng PKML statements between Net-WMS components and databases, using standard tools and protocols for XML not described here. The knowledge models and datasets will be stored in the virtual warehouse in PKML files and in databases. The communication of these data in the directives below is always done in PKML using PackXML exchange format.

We refer to [Deliverable D6.1] for the detailed representation in PKML of items, bins, rules, problem instances, solutions, etc. which are communicated in the following directives.

4.1 Communication from the Virtual Warehouse to the Optimizer

4.1.1 Uploading a Packing Knowledge Model

This amounts to communicate the PKML files defining the knowledge model to the packing solver.

4.1.2 Uploading a Packing Problem Instance

This amounts to communicate the PKML files or statements defining the problem instance.

4.1.3 Solving Request

This amounts to communicate a PKML goal to the packing solver. The goal should involve search predicates, see [Deliverable D6.1 section 4.2.2], in order to compute ground solutions.

4.1.4 Request for other solution

This amounts to communicate a control goal for the PKML solver, which is not a PKML goal properly speaking, but treated as such by a slight abuse of terminology.

4.1.5 Request for Getting a Solution Close to Current Configuration

This amounts to communicate a PKML goal for solving a bin packing problem with the optimization criterion, see [Deliverable D6.1 section 4.2.2], of minimizing the distance to a given configuration represented in PKML.

4.1.6 Placement Requests for one Object

This amounts also to communicate a PKML goal for placing one object only (by labelling its variables).

4.1.7 Placement Requests for One Region

This amounts to communicate particular PKML goals for, e.g.:

- Pushing the objects in one direction,
- Creating a hole in a region,
- Filling in a region,
- Strip packing,
- Increasing/decreasing the weight in a region,
- Etc.

The regions and directions are represented with PKML data structures.

4.1.8 Information Request on one Object or one Region

This amounts to communicate PKML goals for getting numerical values on PKML expressions or domain information on the variables of the problem (the coordinates of the objects, container size, etc.).

It is worth noting that symbolic information requests, such as what are the PKML rules constraining a given object or region, are not supported by this specification.

4.1.9 Control Requests

Control request for clearing the current set of sent PKML statements, or for backtracking in the history of interactions and restoring a consistent state, are assimilated to PKML goals by abuse of terminology. They are transmitted to the constraint program by the PKML compiler and treated with an incremental execution engine described in Section 4.3.

4.2 Communication from the Optimizer to the Virtual Warehouse

4.2.1 Solutions

The solver communicates a representation in PKML of a solution found to a goal (not necessarily a ground solution), i.e. a vector of values or of domains for the unknown variables of the goal.

It is possible that for efficiency reasons when the goal is a large packing problem, the communication should be incremental w.r.t. the last communicated solution. This is not specified in this document.

4.2.2 Failure and Explanations

The solver communicates a notification of failure to answering a request, together with a succinct explanation, e.g. absence of solution, timeout, error, etc.

Explanations involving PKML rules or symbolic causes for failure are excluded from this specification.

4.2.3 Regions

The solver communicates a forbidden/feasible/sure region requested for an object, in the form of a vector of values in PKML.

4.2.4 Strings

None of the requests to the solver that are specified in the previous section uses the communication of a string by the solver. It is given here for provision. For instance, the book-keeping mechanism of the PKML compiler, see [Deliverable D6.1 section 4.3.2], makes it possible for the packing solver to return a PKML statement in the form of a string. This could be used to obtain the PKML rules that apply to an object. Such requests have not been specified however because of the potential difficulties for their implementation in constraint programming solvers.

4.3 Incremental Execution

It is worth noting that all the requests to the packing solver being incremental and backtrackable, the constraint program generated by the PKML compiler [Deliverable D6.1 section 4.3] must be executed with a top level supporting the execution of new goals upon a success, and backtracking to previous interactions. Such execution models have been described in [Carlsson Ottoson Carlson 97, Fages Fowler Sola 95, 98] and are implemented in the constraint component of CLPGUI for instance.

5 Synchronous Communication

As explained in [Deliverable D5.3 section 3.1.5], to be fully functional, the virtual warehouse should have a synchronous connection to the packing solver. By a synchronous connection, what is meant is a synchronization between the physical simulator and the solver in real-time, e.g. with a response time of less than 40ms for maintaining a flow of 25 images per second.

While this is clearly not achievable for all requests to the solver, especially for requests involving optimization or search with many objects, this is nevertheless possible for simpler requests, even in presence of many objects, such as for instance the communication of:

- feasible regions for PKML objects,
- forbidden regions for PKML objects,
- violated constraints
- domain of variables, e.g. for weights, free volume, etc.
- and simple computations involving constraint propagation

Interestingly, the visualization of the results of the solver to these requests raises the issue of their display by Virtual Reality metaphors. However, thanks to the flexible architecture specified in this report for the connection of the virtual warehouse to the packing solver through PKML goals, the proper issue of synchronous communication is purely an issue of response time of the solver (assuming no communication over the network in this context).

6 Conclusion

This report has illustrated and specified the collaborative system between the virtual warehouse users and the packing optimization solver. We have reviewed the main features of this system for:

- uploading packing knowledge models,
- uploading packing problem data,
- getting placement solutions,
- changing configurations in virtual reality and getting close solutions,
- changing the placement rules and visualizing their effect,
- obtaining information on an object or a region.

A running example inspired from the PSA study case [Deliverable 2.2 section 2] has been transcribed in PKML [Deliverable D6.1 section 4], and demonstrated with a proof-of-concept software prototype of the collaborative system in constraint logic programming, by adapting the CLPGUI graphical user interface [Fages Soliman Coolen 04] and using the PKML compiler to SICStus Prolog [Deliverable D6.1 sections 4]. The purpose of this mock-up was to discuss and validate with the partners of the project the specification of the collaborative system between the users and the solver described in this report.

The prototype implementation that will be developed next year for validating the whole tasks made in WP 5 will be based on the graphical user interface developed by the CEA [Deliverable D5.3] and will integrate further requirements of the PSA study case [Deliverable 2.2 section 2.6], of the FIAT study case [Deliverable 2.2 section 3], of the generic placement constraints (WP 4), and of the overall integration in the Net-WMS distributed architecture (WP 8).

7 References

[Carlsson et al 07] M. Carlsson et al. SICStus Prolog User's Manual. Swedish Institute of Computer Science, release 4 edition, 2007. ISBN 91-630-3648-7.

[Carlsson Ottoson Carlson 97] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In H. Glaser, P. Hartel, and H. Kuchen, editors, Programming Languages: Implementations, Logics, and Programming, volume 1292 of LNCS, pages 191–206. Springer, 1997.

[Deliverable D3.1] State of the art of enabling technologies for packing and planning in future WMS. July 2007.

[Deliverable D2.2] Case studies. July 2007.

[Deliverable D6.1] Specification of the knowledge modeller. March 2008.

[Deliverable D5.2] Specification of the collaborative system between user and optimisation module. April 2008.

[Fages Fowler Sola 95] François Fages, Julian Fowler, and Thierry Sola. A reactive constraint logic programming scheme. In Leon Sterling, editor, *Proc. International Conference on Logic Programming ICLP'95*, Tokyo, 1995. MIT Press.

[Fages Fowler Sola 98] François Fages, Julian Fowler, and Thierry Sola. Experiments in reactive constraint logic programming. *Journal of Logic Programming*, 37:1-3:185–212, October 1998.

[Fages Soliman Coolen 04] François Fages, Sylvain Soliman, and Rémi Coolen. CLPGUI: a generic graphical user interface for constraint logic programming. *Journal of Constraints, Special Issue on User-Interaction in Constraint Satisfaction*, 9(4):241–262, October 2004.