Progress on Ptidej

Yann-Gaël Guéhéneuc

File started April 9, 2001
Copy of October 27, 2003
Priorities: 11, 66

Design pattern detection Figure 1 is a summary of the design patterns found in JHotDraw v5.1 and other frameworks. The table shows the design pattern primary participants, if the design pattern is implemented in Ptidej, and if the design pattern has been successfully detected.

Design defect detection Figure 2 is a summary of the design defects found in JHotDraw v5.1 and other frameworks. The table shows the design design defect primary participants, if the design defect is implemented in Ptidej, and if the design defect has been successfully detected.

Entities	**Design Patterns**	**I**	**F**
`util.Command`	Command		
`util.Iconkit`	Singleton		
`standard.NullHandle`	NullObject		
`standard.SelectionTool`	State		
`standard.CompositeFigure`	Composite	✓	✓
`standard.ConnectionHandle`	Prototype		
`standard.ConnectionTool`	Prototype		
`standard.CreationTool`	Prototype		
`standard.DecoratorFigure`	Decorator		
`framework.Locator`	Strategy		
`framework.Painter`	Strategy		
`framework.PointConstrainer`	Strategy		
`framework.Tool`	State		
`standard.AbstractFigure`	Template Method		
`framework.ConnectionFigure`	Strategy, Observer	×, ✓	×, ×
`framework.Connector`	Strategy, Factory Method	×, ✓	×, ✓
`framework.Drawing`	Observer		
`framework.DrawingEditor`	Mediator	✓	✓
`framework.DrawingView`	Observer, State, Strategy		
`framework.Handle`	Adapter		
?	Chain of responsibility	✓	
?	Facade	✓	
?	Iterator	✓	
?	Memento	✓	
?	Proxy	✓	

Figure 1: The **Entities** column contains the names of the entities that are participating in the design patterns given in the **Design Patterns** column. A check mark in the **I** column indicates that the design pattern is implemented in PTIDEJ, a check mark in the **F** column indicates that the design pattern is correctly detected by PTIDEJ.

Entities	**Design Defects**	**I**	**F**
?	?		

Figure 2: The **Entities** column contains the names of the entities that are participating in the design defects given in the **Design Defects** column. A check mark in the **I** column indicates that the design defect is implemented in PTIDEJ, a check mark in the **F** column indicates that the design defect is correctly detected by PTIDEJ.

Fixed bugs and implemented features The following list points out the bugs and missing features in PTIDEJ (including PADL, PTIDEJ SOLVER, ...), and the corrections made. (The list is given in reverse-chronological order).

1. When deploying (installing) PTIDEJ on a new computer, the install program does not update path of the `.ptidej` files. Indeed, it is difficult with INSTALLSHIELD to perform such updates. Maybe the `.ptidej` could be defined both as absolute *and* relative paths, so this problem could be automatically solved (at least) for the examples?

   2003/10/24 ▶ Done! The loading mechanism now handle relative paths.

2. A nasty bug was hiding in the `PropertyManager` class. The installation could not work on computer other than mine because of the hard-coded full paths.

   2003/10/24 ▶ The `PropertyManager` now uses the path of its client class as a base from which to load other resources. This should give much more flexibility to the installation and also to the call to the constraint solver. This should also ease the implementation of the Ptidej plug-in for Eclipse.

3. Rename the packages to replace *metamodel* with *pdl* (or *padl*), to match the package names in CAFFEINE.

   2003/08/20 ▶ Done!

4. Improve error handling when an extension requires missing classes.

   2003/08/06 ▶ The constructor of `jtu.ui.extension.Repository` now catches `NoClassDefFoundError`.

5. Unify the frame and panel dimension in `Viewer` and `ViewerPanel`!

   2003/06/06 ▶ The dimension declared in the `ViewerPanel` was actually useless, so removed.

6. When displaying a solution, the entities of the concrete pattern are not properly connected. For instance, the `CompositeFigure` class and the `Figure` interface are not connected when displaying a solution containing only these entities (not the `AbstractFigure` class).

   2003/04/04 ▶ The problem was that the re-connection algorithms were based on the pattern model, not on the "underlying" abstract-level model.

7. In the ECLIPSE plug-in, the display associated with the primitive factory known by the control is sometimes out-of-sync. with the diagram editor, which causes "Widget disposed" error messages.

   2003/01/30 ▶ I had forgotten to implement the `DiagramEditor.dispose()` method to detach the listeners created along with each instance of the `DiagramEditor` class. Thus, when closing a `DiagramEditor`, its *ghost* would still be attached to the `PreferenceStore`, causing the exception.

8. The methods of `SolutionBuilder` are not consistent. Some methods take an instance of `Model` as parameter, others take an instance of `Container`.

   2003/01/30 ▶ The paramters are now consistent: The public methods take either `Entity` or `Property` or `AbstractModel` instances as parameters. The `getSolutionPattern(Solution, AbstractModel)` builds an instance of class `PatternModel`.

9. In the `SimpleDocument` of the demo applet, the name of the classes should be made consistent and... proper english!

   2003/01/15 ▶ Done.

10. The preference panel of the Eclipse plug-in, the control view, and the diagram editor should be connected so that when one changes, the other ones reflect the changes.

    2002/12/27 ▶ The `ControlView` and the `DiagramEditor` now both uses the preference store to set and to get visibility values, which keep them all in sync.

11. I need to add an extension mechanism for Ptidej: I want to call the adjency matrix display from the OADymPPaC projet (Mohammad Ghoniem), but I don't want to hack the viewer, I need something more robust that could also work for the `ConstraintResultsFrame` and in the Eclipse plug-in as well!

    2002/12/21 ▶ I added a whole set of classes: `Extension` is an interface that any Ptidej extension must implement; `Repository` is a singleton class that manages the extensions. I now use this mechanism to make a link with OADymPPaC! The replacement of `ConstraintResultsFrame` should be done soon...

12. The algorithms of the `*Inheritance*` constraints were buggy! But the bugs did not show up with Claire 2... I was enumerating through the entities only if they had at least one super-entity!

    2002/12/21 ▶ Done. I fix the bugs up! (Well, I have more confidence with Claire now... Thank also to the JUnit tests!)

13. The display would be much nicer if we could show/hide assocation, aggregation, and composition relationships as well as ghost entities, all independently!

    2002/12/15 ▶ Done! And it's *way* better!

14. It should be possible to load a single class-file at a time in addition to complete directories and jar file.

    2002/12/15 ▶ A `addFile()` method has been added to the `ProjectViewerPanel`.

15. In the constraints and constraint tests, I should replace the word "class" with "entity" to be consistent.

    2002/12/3 ▶ The names are now consistent.

16. It is possible to create a Ptidej project with no name (or a blank name).

    2002/10/12 ▶ The `newproject()` method now handles blank name by *not* creating project when a blank name is provided.

17. The `ContainerAggregation.recognize(...)` method is not flexible enough. For example, in JUnit, it does not recognize some container-aggregation relationships because of missing `remove(...)` method or different list of argument. I need to introduce a mean to handle those extra-cases.

    2002/10/12 ▶ It is now possible to load (after loading a project) a specific `.pdl` file that specifies some extra relationships among classes and some modifications to the current model. Using this mechanism, it is possible to add extra association, aggregation, or composition relationships and to convert aggregation relationships into composition relationships.

18. Improve the Ptidej–Caffeine integration by specifying the content of *Caffeine* files (`.caffeine`), which contain the results of the extraction of behavioral information.

    2002/10/12 ▶ The integration between Ptidej and Caffeine is now based on the mechanism to add extra information to a model using `.pdl` files.

19. When building the graphic entity, all knowledge and binary class relationships are not displayed.

    2002/09/18 ▶ I was using the `String.indexOf(String)` method without taking care of the end-of-line character. If the text already contained `TestListener\n` and I added `Test`, `Test` would not be added because it is already present... I now test for the presence of `Test\n`

20. An `ArrayIndexOutOfBoundException` occurs in the `InheritanceClusterLayout` algorithm, when displaying concrete patterns which handle interfaces.

    2002/08/05 ▶ The problem came (again!) from the interface-management algorithm.

21. The pattern visitors and listeners must distinguish between `Aggregation` and `Composition` relationships and `ContainerAggregation` and `ContainerComposition` relationships.

    2002/08/05 ▶ All meta-model dependent visitors and listeners are now up-to-date.

22. For size complexity, replace the AC4 ignorance constraint by constraints opposed to the knowledge, association, aggregation, composition (awareness) constraints.

    2002/08/23 ▶ I try out this solution, however the solutions found are different even though the opposed constraints strictly declare the complementary set of couples of the *normal* ignorance constraint.

    2002/08/30 ▶ We fix the problem with Narendra Jussien: When I created the *not feasible*-AC4 constraint, the list of supports was not built correctly and was actually "as-if" it was a *feasible* constraint.

23. When loading the JDT/Core, the recognition attempts to add some container-aggregations twice.

    2002/08/29 ▶ The problem was only that I did not append a unique identifier to the name of the container-aggregation relationships. When a class implements more than one container-aggregation relationship, the unique identifier distinguishes them.

24. The recognition algorithm of class `PatternIntrospector` increase the number of entities to recognize as projects are loaded.

    2002/08/29 ▶ The bug was in fact in the `ProjectLoaderViewerPanel.newProject()` method, which added a new instance of `PatternStatitics` to the `ListenerManager` each time a new project was created.

25. The `InheritanceClusterLayout` algorithm still has some problems with `Ghost` entities. It throws an exception when loading the JDT/Core plug-in.

    2002/08/29 ▶ The problem actually came from the meta-model: When recognizing an interface, I wasn't adding its ghost super-entities into the model. So, this interface had a inheritance-depth of 0, as a ghost entity.

26. When building a pattern from source files, the algorithm connecting classes and interfaces fails to connect classes with their super-interfaces since the introduction of the `Ghost` entity.

    2002/08/28 ▶ The problem came from the ghost entities. When recognizing an entity, say a class, the algorithm created a ghost entity matching the super-class of the recognized entity. Then, the real super-class was analyzed and it tried to update the first entity: The update failed because an actor (the ghost entity) with the same actor ID already existed! From now on, before adding a new super-entity (class, interface, or ghost), I make sure to remove any other entity with the same actor ID. This is sufficient because the being-added super-entity is necessarily more *real* than the potential existing ghost entity.

27. The constraint-relaxation mechanism does not seem to perform what it is supposed to: When a constraint is relaxed, the constraint should be replaced by a related constraint of lesser weight.

    2002/08/27 ▶ The `CombinatorialAutomatiSolver` performs the computation of the solutions using both a combinatorial algorithm and a constraint-replacement algorithm. The tests on the Composite design pattern seem to valid its implementation (`jtu.solution.test.examples` package).

28. Distinguish among problems and solvers. Problems may be `CUSTOM` or `AC4`, solvers may be `AUTOMATIC`, `COMBINATORIAL_AUTOMATIC`, or `SIMPLE_AUTOMATIC`.

    2002/08/27 ▶ The `ConstraintViewerPanel` now offers five different checkboxes, which allow the different valid combinations of type of problem and of constraint solvers.

29. Distinguish between the `Aggregation` and `Composition` relationships and some new `ContainerAggregation` and `ContainerComposition` relationships.

    2002/08/25 ▶ I added two new elements to the meta-model: `ContainerAggregation` and `ContainerComposition`. I also added graphic constituents to distinguish among them. The symbols now are:

Knowledge	:	`-k-->`
Creation	:	`-*-->`
Association	:	`---->`
Aggregation	:	`[]-->`
Container aggregation	:	`<>-->`
Composition	:	`[#]->`
Container Composition	:	`<#>->`
Specialization	:	`-|>-`
Implementation	:	`-|>-`

30. I need to connect the interfaces with the `java.lang.Object` class if they have no other super-interfaces. (This is a simple bug-fix in PatternsBox, however, this bug-fix introduces a bug in the layout manager...)

    2002/08/24 ▶ This was a nasty bug in the `InheritanceClusterLayout` class. I was moving interfaces all the way up in the display hierarchy, then, when creating the tree-nodes, I could not find the super-entity of the interfaces (`java.lang.Object`) because it had not been added yet! Fortunately, the bug was very easy to fix... See class `InheritanceClusterLayout`.

31. The refactoring in PatternsBox introduced a bug when linking a `PClass` with its super-`PInterfaces`.

2002/08/23 ▶ I had broken the connection mechanism between a class and its super-interfaces in the `PClass.recognize(...)` method. Now it works fine. (I added a JUnit test.)

32. The domain generated by the different `PaLMDomainGenerator`s does not distinguish among knowledge, association, and aggregation (of cardinality 1) relationships: These three relationships are enforced by the `makeKnowledgeAC4Constraint` constraint. I need to distinguish these relationships to improve design defect detection such as the Redundant Transitivity pattern.

    2002/08/22 ▶ The `PaLMAC4DomainGenerator` now provides different constraint for the knowledge, association, aggregation, and composition relationships. The different detection algorithm have been rewritten to use these constraints. I also implemented unit tests for the Ptidej constraint solver.

33. The computation of the message types (in class `PLink`) needs some heavy refactoring, with regards to Stéphane Ducasse *et al.*'s book (see pattern 10.3) and to the corresponding unit tests.

    2002/08/22 ▶ I first re-implemented the algorithm using pattern 10.3. However, the problem was much more complex with regards to bytecode analysis, so I implemented a real stack-based bytecode analyzer, which is much robust and accurate. I also implemented several unit tests dedicated to the `PatternIntrospector` class.

34. Find out why the computation of the message links loops over and over again on the same entities.

    2002/08/22 ▶ The problem was that the relationship recognition mechanism was not included as a `PatternElement` in the reverse-engineering algorithm (in `PatternIntrospector`) but was called by the different elements (such as `PAssociation`, `PAggregation`, ...) and thus was called multiple times. From now on, the `PLink` class is included in the reverse-engineering algorithm and the other elements do not `super`-call the relationship recognition mechanism.

35. The domain generated by the `PaLMAC4DomainGenerator` is too big, for example for JUnit, with regards to the `makeNotEqualAC4Constraint` and `makeEqualAC4Constraint`.

    2002/08/14 ▶ I now use constraints in "intension" (such as in the custom algorithm) to limit the size of the generated domain.

36. Rewrite the constraints to use the AC-4 algorithm with explanations.

    2002/08/13 ▶ All pattern constraint declarations now possess their AC-4 with explanations counterparts. Also, I implemented a set of JUnit tests to check the results of the constraints.

37. Constructors declaration are not handled by PatternsBox.

    2002/08/13 ▶ Constructors are now analyzed by PatternsBox.

38. The definition of the interface for `jtu.ui.primitive` must be made consistent: All the methods of the abstract factory must accept a parameter `RGB` color.

    2002/08/07 ▶ Done.

39. The `MultiRepresentation` mechanism must be reviewed and cleaned up. Especially, with regards to redundant information about the instances of `ClassLoader` and of the instances of `ClassLoader` known by the instances of `PatternIntrospector`.

    2002/08/07 ▶ Instances of the `MultiRepresentation` are using a Method factory pattern.

40. When loading the class files (from a JAR) of the JDT/Core plug-in, PatternsBox and Ptidej eats away more than 150Mb! Need to improve memory footprint!

    2002/08/06 ▶ I used JProfiler from EJ-Technologies, it is great! I analyzed the analysis and display of large framework and found several places for improvements. From now on, it should be possible to load in Ptidej frameworks up to thousands of classes and interfaces and ten-thousands of relations. Correcting this limitation involved numerous refactoring and corrections, PatternsBox and Ptidej are in better shape than ever now (but bug-free code does not exist...).

41. Need to refactor the method `ConstraintResultsFrame.build()`.

    2002/08/06 ▶ Done.

42. When loading files and the path does not exist (as in a project file), the `TypeLoader.loadSubtypesOfFromDirInto(...)` method just throws a null-pointer exception: This behavior must be improved!

    2002/08/06 ▶ The problem was the wrong test that only checked the length of the list of files, not its nullity.

43. When concrete patterns are loaded, the group solution tip (or group solution pattern) is built once and for all, using the element visibility of the source code model at the time of the loading. If the visibility changes, the group solution pattern is still displayed the old way.

    2002/08/05 ▶ From now on, changing the visibility of the code also changes the visibility of the group solution patterns.

44. The visibility of the solution patterns should be modified according to the visibility associated with the source code. (The visibility of the solution patterns should not be set once and for all at the loading of the concrete patterns.)

2002/08/05 ▶ Done. (See previous bug.)

45. Cloning a subset of the pattern requires cloning the whole pattern, which is memory-consuming. Need to implement a better algorithm. See methods `PEntity.performCloneSession()` and `PClass.performCloneSession()`.

    2002/08/04 ▶ The cloning of a subset of a pattern now follows the cloning protocol.

46. There is a bug when building the solution pattern for the Composite design pattern on `jtu.tests.composite2`. The first time the solution pattern is built, the class `ParaIndent` is not connected with the class `Paragraph`. Only the second time that the solution pattern is displayed.

    2002/08/04 ▶ Now that the cloning protocol is strictly followed, the problem is fixed.

47. For the moment, everything is based on the reflection API of Java. However, using the reflection API presents several shortcomings. First, we load class through an instance of the class `ClassLoader`, thus the `ClassVerifier` of the JVM processes each class and the JVM executes its static initializers, which are both useless. Second, each class must be loaded along with all its dependent classes, which prevents us to analyze subset of large framework, such as ECLIPSE.

    2002/07/28 ▶ I suppressed all reflection-related mechanisms in the analysis of classes. From now on, classes and interfaces are represented as instances of class `ClassFile` from CFPARSE.

48. ACRVIEWERPANEL does not memorize all the packages that have been loaded: This is a problem when creating the `MainRunner`.

    2002/07/28 ▶ I now use instances of `ClassFile` to represent class files. I also memorize the list of paths and jar files from which I loaded class files.

49. Improve the algorithm related to the main path and main package in the class `ACRViewerPanel`.

    2002/07/28 ▶ Done.

50. Implement `New project`, `Load project`, and `Save project` algorithms.

    2002/07/28 ▶ Done.

51. The method `ACRViewerPanel.setCurrentPackageAndPath(String)` needs some heavy refactoring!

    2002/07/28 ▶ Done. This method is not needed anymore.

52. Allow loading dynamic information from a random file.

    2002/04/08 ▶ The `Load dynamic information` feature now possesses two behaviors: First, it loads the properties of the selected constituents; Second, if no constituent is selected, it opens a dialog to select of file.

    2002/07/04 ▶ Lofti Hazem found a bug when loading dynamic information from the example files. I had changed the specification of the dynamic information file without modifying the `ACRViewerPanel.loadDynamicInformation(DEntity, Properties)` method. I also modified the documentation accordingly.

53. The layout is not *that* great ...

    2001/07/28 ▶ Two new layouts are now available. The latest (and more efficient) one sorts entities according to their depth in the inheritance tree, and then move the subclasses as close as possible to their superclass by *waves*; i.e., one inheritance level at a time, ensuring a better looking layout with as few inheritance (superclass) links crossing as possible.

    2002/07/01 ▶ The new `InheritanceClusterLayout` makes sure the class-hierarchy tree is cleanly displayed and displays interfaces first.

54. Improve and test the creation of builds with InstallShield.

    2002/07/01 ▶ Done. Thanks to Eclipse [2] capabilities, I manage from the very same workspace the projects, their source files and class files, and the InstallShield files. Thus, whenever I build a new version of Ptidej, InstallShield always uses the latest class files and resource files.

55. When changing the view from one model to another, the associated instance of `ClassLoader` is not changed. This prevents from adding different packages in different projects in various order.

    2001/04/20 ▶ This is now corrected: The `TriRepresentation` class is renamed `MultiRepresentation` and it can hold any kind of specific values (in a hashtable). The instances of `ClassLoader` and `PatternIntrospector` are now set according to the model currently displayed. The system complexity, when calculated, also changes according to the current model displayed. However, the `MultiRepresentation` must be improved.

    2002/04/03 ▶ There is now a real `MultiRepresentation` class, which contains the data per view (pattern, complexity, solutions, ...) and which can be extended. This new representation also helped me in improving the complexity computation and display mechanism.

56. The call to the constraint solver is still not working correctly. Should try without the batch file.

    2002/04/03 ▶ This bug was hard to fix! There were several problems counteracting this single effort:

- The Claire code for the constraint solver needed some clean-up. There is now one and only one global variable, `PtidejResourceDir`, which contains the directory where the resource file is to be found. This variable is set by PTIDEJ UI in the instruction file (`Instructions.cl`), file given as parameter to the constraint solver (`Ptidej.exe`). This global variable must be removed in the future.
- The instruction file is now unique, whether PTIDEJ SOLVER is interpreted or compiled, it uses the same file.
- The resource file did not contain a Claire-compliant path for the PTIDEJ SOLVER resource directory. The path must not be: `"C:/Documents and Settings/Yann/Work/Ptidej Solver/Ptidej/"`, but: `"C:/Docume~1/Yann/Work/Ptidej~1/Ptidej/"`.
- The constraint solver looped when searching the Composite on the framework JUNIT v3.7.

2002/04/07 ► Now that this bug is fixed, the automated tests work perfectly, thanks to JUNIT v3.7.

57. When calling the constraint solver, get the `stdout` of the process and display it as a feedback for the user.

    2002/04/07 ► Using the `OutputMonitor` from CAFFEINE, I now display the output from the constraint solver in the console.

58. Add a `PatternListener` mechanism to listen to the construction of patterns.

    2002/04/02 ► Done. I need to clean up the `PatternProperty` class though, to make sure I do not duplicate the field `Vector listOfPatternListeners` inappropriately. Also, the method `Pattern.removePEntity(PEntity)` does not detach the listeners of the removed entities. (There is a similar problem with the elements.)

59. Rename the class `PatternRootElement` into `PatternConstituentRoot`.

    2002/04/02 ► Done. However, a better name (to discuss with Hervé Albin-Amiot) could be `PatternConstituent`.

60. The solutions found after a constraint has been relaxed are not quite what they seem: So far, constraint are cumulatively removed from the problem at hand. What we need is that a constraint is removed, the solutions to the problem are calculated, then this constraint is re-added and another constraint is removed, and so on and so forth.

    2001/11/27 ► Using the latest PALM version (v1.07), a combinatorial constraint solver algorithm has been implemented. This solver algorithm remove all the constraints in a combinatorial fashion. Thus, all the possible combination of constraints are explored.

61. When creating the solutions, the patterns associated with each of the solution are created up-front, which is time- and (very) memory-consuming.

    2001/08/30 ▶ The `SolutionBuilder` provides two methods: First, `getSolutions()`, which builds the solutions from the constraints and a source code; Second, `getSolutionPattern()`, which builds a solution pattern from a solution. The method `SolutionBuilder.getSolutionPattern()` is called in `GroupSolution.setSelected()` when (and only when) needed.

62. Refactor the graphic framework kernel and primitives to implement an abstract factory and factorize graphic-framework independent mechanism from the AWT implementation to reuse in the Eclipse implementation. This must allow to have different implementation of the primitives (for example, AWT and SWT).

    2001/08/15 ▶ Done. The project `Ptidej UI` defines a package `jtu.ui.primitive` that defines a set of interfaces for the graphic widgets. The implementation-specific classes are located in the `Ptidej UI Primitives AWT` and `Ptidej UI Primitives SWT` projects.

63. Because Swt does not have a `GC.translate()` *à la* AWT, need to add a pair of offset coordinates to each `paint()` method of the graphic kernel and of the graphic primitives.

    2001/08/13 ▶ Done.

64. When loading or adding a package, only class files should be displayed by the file dialog.

    2001/08/12 ▶ It seems that there is a bug in the class `FileDialog`: The instance of the class implementing `FilenameFilter` is never used. To filter out the file names that have not the desired extension, I use the method `FileDialog.setFile()`; eg., `fileDialog.setFile("*.ptidej");`.

    2001/08/10 ▶ The good solution is actually to provide the instance of class `patterns.util.ClassLoader` (used to load the `Entity`, `Element`… classes) with the instance of the current class loader (used to load the `PatternIntrospector` class), as parent. The `Method.invoke()` method checks the parent–child relationships and does not complain, because it is acceptable.

65. The `XCommand` filed must be broken into multiple lines, and the solution tip must display each of its lines with the correct indentation.

    2001/08/04 (IJCAI 2001, Seattle) ▶ The mechanism to break the `XCommand` on multiple lines was there. Only the definitions of the constraints needed to include $\backslash n$ and $\backslash t$ in their `XCommand` declarations.

66. The print feature does not take into account the visibility settings from the `ViewerPanel`, by default, there are no visible elements.

2001/07/31 ▶ The visibility is now the same as the current source code.

67. The layout algorithm should be based on *pass* according to the depth on the inheritance lineage of the entities.

    2001/07/30 ▶ The new layout `InheritanceDepthLayout3` implements an algorithm based on inheritance depth and on passes.

68. The pattern displayed as tip for a (complete or distorted) solution has no visibility set (thus, no graphical elements or names were displayed, beside the entity names).

    2001/07/30 ▶ The solution pattern (i.e., the concrete patterns) has the same visibility as the corresponding source code at the moment the solutions are loaded.

69. When showing a `GroupSolutionTip` from the `ConstraintResultsFrame`, the size of the `Canvas` is not re-calculated.

    2001/07/30 ▶ The instance of `ConstraintResultsActionListener` defined in method `MACRViewerPanel.displaySolutions()` takes care of adjusting the size of the canvas.

70. When changing the source displayed, the visibility of the entities and elements of the new source being displayed is not updated.

    2001/07/25 ▶ The class `MACRViewerPanel` now implements a method `displayCurrentPattern()`, which takes care of setting the appropriate visibility on the source code being displayed.

71. The communication mechanism between the `ViewerPanel` and any extension (for instance, `ConstraintResultsFrame`) is too primitive. Need to add an observer pattern (define an event and a listener interface).

    2001/07/25 ▶ The `ViewerPanel` now defines a listener `SourceListener` and an event `SourceEvent`.

72. Improve the communication between the `ConstraintResultsFrame` and the `MACRViewerPanel` classes. Maybe a notification from the source choice combo-box would be sufficient?

    2001/07/25 ▶ The methods dealing with the source code being displayed in the `ViewerPanel` (such as `ViewerPanel.addPattern()`, `ACRViewerPanel.loadConstraintsInformation()`, `ACRViewerPanel.loadDynamicInformation()`, and `ACRViewerPanel.removeAllConcretePatterns()`) take care of sending the appropriate instance of `SourceEvent` to the instances registered as `SourceEventListener`.

73. The concrete pattern displayed with a chosen solution is incorrect.

2001/07/23 ▶ The problem was that the solutions were sorted after *the solution patterns were built, thus leading to inconsistencies between the array of solutions and the array of solution patterns.*

74. The number of links found in the class `jtu.tests.mediator2.Mediator` is wrong.

    2001/07/23 ▶ The problem was in the method `PLink.recognize()`. The algorithm matching the method at hand with the corresponding method information (extracted by CFParse from the class file) was based solely on the method name. In the case of the class `jtu.tests.mediator2.Mediator`, the methods have the same name operation *but different parameters. The algorithm now checks the method complete signature.*

75. Improve the loading of files to remove the unnecessary selection of the main directory.

    2001/04/23 ▶ From now on, loading or adding a package consists in selecting a single class file within the package to load.

    2001/07/22 ▶ The notion of project has been added to ease the loading of files

76. Abstract from `ACRViewerPanel` the algorithm to build solutions.

    2001/07/22 ▶ The algorithms to generate constraint results (i.e., to solve the CSP) and to build the solutions from the results are now located in the `SolutionBuilder` class.

77. The field `listOfClasses` contains only `null` elements when creating an instance of class `PatternIntrospector`, this should not be so: To be verified.

    2001/07/21 ▶ The problem was related to the new representation in the `ViewerPanel`: A class loader was created, but a different class loader was used to load the class files from the instance of `PatternIntrospector`. Now, the correct (and corresponding) instance of `ClassLoader` is used along with the instance of `PatternIntrospector`.

78. When lots of entities are loaded, it is difficult to find the one looked for. Need to add a *goto* function that would localize the entities and focus the window on it.

    2001/07/20 ▶ A new mechanism has been introduced. The `ViewerPanel` defines two methods `addSourceListener()` and `removeSourceListener()` to notify instances of any class that want to know when the source code loaded and shown changes. This mechanism is used to implement the `ConstraintResultsFrame`. The `ConstraintResultsFrame` displays the constraint results associated with a given source code, and allows to show/hide results, and to focus on a specific actor. (This `ConstraintResultsFrame` also provides a listener `ConstraintResultsActionListener`.

79. The display of solutions must be improved: First, the pop-up comment displayed is taken in a random order, a tab-like mechanism should be added; Second, when a solution contains lots of entities, it is difficult to grasp what entities belong to the solution and their relationships.

    2001/07/20 ▶ This problem is partially solved with the `ConstraintResultsFrame`. Still need to implement a tab-like mechanism, though.

80. The solutions built from the constraint results contain redundant and useless information: When a solution at 100% exists, distorted solutions with the same actors also exist ("Qui peut le plus, peu le moins"), these distorted solutions constitute *noise* and must be removed.

    2001/07/19 ▶ The method `SolutionBuilder.getSolutions()` now includes an algorithm to remove distorted solutions that are equivalent to a complete solutions.

81. Track and fix the problem of resources sharing the same name between the projects PatternsBox and Ptidej. Refactor the class `PropertiesManager` and make clear where and how the resources should be accessed.

    2001/10/19 ▶ Done. The problem came from my mis-understanding of the use of the `getResourceAsStream()` method. There is a difference between:

    - `clientClass.getResourceAsStream(...).`
    - `clientClass.getClassLoader().getResourceAsStream(...).`

    The former looks for the resource in the directory corresponding to the class path of the `clientClass`, while the latter looks for the resource in the common directory, corresponding to the class path of the instance of `ClassLoader`.

82. A button to remove at once all the concrete patterns should be implemented!

    2001/07/18 ▶ Done.

83. Remove method `getClasspath()` from the abstract models in the repository.

    2001/07/17 ▶ The `recognize()` methods no longer take an instance of `Pattern` as argument, but an instance of `PatternIntrospector`. This modification makes more sense and makes the code cleaner. The class path is now set within the instance of class `PatternIntrospector` and is queried from it.

84. The algorithm of structure modification in `ACRViewerPanel.modifyStructure()` did remove only the related groups with same percentage and parameter names and values. This let a lot of *noise*.

    2001/07/17 ▶ The noise is now gone: Groups are now related only by parameter names and values.

85. The font used in the buttons for AWT is too big. And the button constructor and display algorithms are too rigid.

    2001/07/16 ▶ If available, the font used now is Tahoma 9pt, and the button algorithms support any font and size.

86. Clean up the listener / `actionPerformed()` mechanism.

    2001/07/15 ▶ Thanks to Hervé Albin-Amiot, who explained me all what must be known on listeners, there is now no more useless or dubious `actionPerformed()` calls.

87. For an unknown reason, the constraint results for the Composite pattern possess one solution at 100%.

    2001/06/30 ▶ This problem is resolved using the new 1.0 versions of Choco and PaLM.

88. The method `PEntity.listInherits()` should (maybe) return a clone of the `inherits` vector, to prevent unwilling modifications.

    2001/06/18 ▶ Done.

89. Because of the methods `clone()` defined on class `Pattern`, classes `PEntity` and `PElement` do not update correctly the links among cloned `PElements` and cloned `PEntities`.

    2001/06/18 ▶ The meta-model now implements an extended cloning mechanism. The new mechanism includes a three-step cloning algorithm: `startCloneSession()`, `performCloneSession()`, and `endCloneSession()`. These methods return `void`, to use them, see `Pattern.clone()`. The idea is that a pattern may be cloned, but a constituent of a pattern (`PElement` or `PEntity`) may not be cloned individually: It does not make sense to clone a `PMethod` if the rest of the pattern is not cloned too. It is actually a risk: To clone a single constituent on an individual basis may prove to create duplicates with references on old objects... Thus, only the `Pattern` class implements the `Cloneable` interface. The constituents of the pattern implements the `CloneablePatternConstituent` interface. This interface provides three methods: `startCloneSession()` is somewhat equivalent to a shallow copy of the constituent. After this protocol is executed, all constituents are guaranteed to be shallow-copied. No assumption is made about the links among constituents. `performCloneSession()` updates the links among constituents, using the `isCloned()` and `getClone()` methods. After the execution of this protocol, all the links are guaranteed to be up-to-date, somewhat like a deep copy. `endCloneSession()` finishes the updates and cleans the possible temporary values, mainly it sets to `null` all the `clone` instance variables.

90. Complete `PLinkingMethod.computeObjectReferencesNames(...)`.

    2001/06/18 ▶ The whole link management mechanism has changed. Two reasons: First, to factor code in the hierarchy; Second, to improve the detection of associations, aggregations, and compositions, and to make the algorithms more flexible.

91. Distinguish visuals among creation links, reference links, delegation links, association, aggregation and composition.

    2001/05/06 ▶ The code is:

Knowledge :	`-k-->`
Creation :	`-*-->`
Association :	`---->`
Aggregation :	`<>-->`
Composition :	`<#>->`
Specialization :	`-|>-`
Implementation :	`-|>-`

92. Allow a class to have more than one composition. (Create a list of list and handle within the constraints system.)

    2001/05/06 ▶ The property `componentsType` is a list of `PEntity` and the property `components` is a list of list of `PEntity`.

93. The method `DHierarchy.toString()` prints the symbol `<?>->` instead of `-|>-`.

    2001/05/06 ▶ This is now corrected: The `DGraphicalElement` has been refactored, the `toString()` method is defined into class `DGraphicalElement` and calls the `getName()` method.

94. The current meta-model is not fuzzy enough to handle the detection of distorted implementations of associations, aggregations, and compositions. For exemple, if the field holding the reference appears to be `protected`, no association would be built.

    2001/05/01 ▶ The modification of the hierarchy of `PElement` and the addition of a new algorithm (`PLink.recognize()`) partially (if not totally?) address this problem. This solution needs a careful review and additional testing.

95. The method `PLink.computeMessageTypes()` does not correctly handle the cardinality of return type and parameters. This method needs a serious re-factoring.

    2001/05/01 ▶ The method has been re-factored: Tons of modifications! It seems to work fine, but it needs to be reviewed by Hervé Albin-Amiot, and to be tested on more cases.

96. Put all the buttons in a scrolling panel.

    2001/04/23 ▶ The panel containing the buttons is now embedded into an instance of the `ScrollPane` class.

97. In ACRViewerPanel, replace text- and button-creation mechanisms with reflection.

    2001/04/23 ▶ The `ViewerPanel` class handles the addition of new widgets, buttons, or separators.

98. There is a problem of uniqueness among constituents of a `DPattern`. When cloning a pattern and building a new `DPattern`, some links are kept on non-existent `DEntities`. This problem comes from the `static` field `DPatternRootElement.backwardLinkPtoD`. This field must be removed.

    2001/04/23 ▶ The static field has been removed. Now, instances of `DEntity` point to their enclosing instance of `DPattern`. An instance of `DPattern` knows all its instances of `DEntity`. Along with this modification, some methods have been removed (`getSuper()` and `hasSuper()`, among others) to simplify the framework.

99. The Ptidej Solver meta-model does not differentiate between knowledge links and creation links. The distinction could improve the detection of the Factory Method pattern.

    2001/04/20 ▶ The Ptidej Solver meta-model has been enhanced. The meta-model now distinguishes among `superEntities`, `componentsType`, `components`, `knownEntities`, `unknownEntities`, and `createdEntities`.

100. The `InheritanceConstraint` does not correctly handle all the possible solutions.

    2001/04/20 ▶ This is now corrected: The methods `awakeOnRemove()` and `awakeOnRestoreVal()` call directly `awakeOnEnum()`. This is very inefficient, but it will stay so until I meet with Narendra Jussien.

101. The relations of knowledge[1] are not built among interfaces. For example, interface `framework.Figure` possesses a method with `framework.Connector` for return type, but no knowledge link is built

[1] An entity `A` knows another entity `B` if `A` possesses a method that refers to `B` (in the method body, as return type, ...). This relation is called knowledge link or acquaintance in [1].

between these interfaces. If the knowledge link existed, Factory Method concrete patterns could be detected.

2001/04/19 ▶ This is now corrected: Instances of `PLinkingMethod` are built for interfaces, and the algorithm manages return types and parameter types. The algorithm also distinguishes among primitive types.

102. Factory Method pattern. The `InheritanceConstraint` and `StrictInheritanceConstraint` only deal with direct super-classes. These constraints must handle super-interfaces as well, to improve the results of the detection of the Factory Method pattern.

    2001/04/19 ▶ The `InheritancePathConstraint` deals with direct super-entities and super-entities all the way up to `java.lang.Object`.

103. Method `addPackage()` does not allow to load packages from different root directories.

    2001/04/18 ▶ The management of the `ClassLoader` has been improved: One instance of `ClassLoader` is created with `Load package`. This instance of `ClassLoader` is used for the following `Add package` method calls.

104. The `InheritancePathConstraint` does not correctly handles all the possible solutions.

    2001/04/14 ▶ The constraint now gives all the correct answers (and only them).

Bugs and missing features The following list points out the bugs and missing features in PTIDEJ (including PADL, PTIDEJ SOLVER, ...).

1. The interface `framework.DrawingEditor` participates *indirectly* to the Mediator pattern (because `framework.DrawingEditor` is an interface) through the class `standard.DrawApplication` and class `standard.DrawApplication` inner classes. Inner classes should not appear in the PTIDEJ SOLVER meta-model, the relations they define should be included in the enclosing class.

2. The interface `framework.DrawingEditor` participates *indirectly* to the Mediator pattern, but it should be included into the results because the *real* mediator object, `framework.DrawingEditor`, implements it.

3. Singleton pattern. Looking at its implementation, the `util.Iconkit` class does not have a private constructor: An instance of the class `util.Iconkit` is created when initializing the application. This instance is stored in a field but never accessed again. All other accesses to `util.Iconkit` are realized through the static method `Iconkit.instance()`. The PTIDEJ SOLVER meta-model should include method-level information (such as method kinds (constructor, method), method modifiers, method name?, ...).

4. State and Prototype patterns. The PTIDEJ SOLVER meta-model should include, maybe, method-body information.

5. Factory Method and Template Method patterns. The PTIDEJ SOLVER system should include a mechanism of automatic *fall-back*. A mechanism of fall-back would allow the relaxation of a constraint by removing this constraint and by adding a new and related constraint. For example, if the `CompositionConstraint` cannot be satisfied, this constraint should be removed and replaced by an `AggregationConstraint`, which could be itself replaced by an `AssociationConstraint` constraint[2].

6. Observer pattern. The Observer pattern implemented by the `Drawing` and `DrawingChangeListener` classes is distorted. The `StandardDrawing` class notifies its observer, `StandardDrawingView`, through the `DrawingChangeListener` interface, but the `StandardDrawingView` class does not *directly* query the new state from the `StandardDrawing` class. The `StandardDrawingView` class uses the `DrawApplication` class as an intermediary. (The `DrawApplication` class aggregates an instance of `StandardDrawingView` and knows about `StandardDrawing`.)

7. Memorize within the meta-model the entities part of an association, an aggregation, or a composition not just the superclass of them all.

8. Improve printing (reduce diagram size if possible).

9. When printing, for some reasons, the font used is bigger than the one displayed on screen.

10. Compute dynamic links.

11. In class `Pattern`, problem of duality between actor and `PEntity`. The problem is even deeper than that: As discussed with Hervé Albin-Amiot, it appears that there is a missing identifier in PATTERNSBOX. For example, in the Composite pattern: The actors `Composite` and `Component` are *fixed*, while actors of type `Leaf` may be added; i.e., in the abstract model of the Composite pattern, an actor `ZLeaf` may be added, using methods `addLeaf()` and `removeLeaf()`. But, when walking the list of actors using the `getActorID()` method, the actors found are:

    - `Composite`
    - `Component`
    - `Leaf`
    - `ZLeaf`

    In *theory*, actors `Leaf` and `ZLeaf` are the same. This is an important point when it comes to deal with those two actors using the available properties

[2]The `AssociationConstraint` and the `RelatedClassesConstraint` are identical.

related methods of the patterns; i.e., `addLeaf()` and `removeLeaf()`: Indeed using the available methods, it is impossible to act on `ZLeaf`. Thus, we must introduce three levels of identifiers:

- At the meta-model level, `getActorType()` identify a unique type of actor. For example, `Composite` or `Leaf`. The methods related to the properties of an abstract model (such as `addLeaf()` and `removeLeaf()`), discovered using the introspection mechanism, work at this level: They work on the type of the actors.
- At the abstract-model level, `getActorID()` distinguishes among actors of same or of different types. For example, `Composite`, `Leaf`, and `ZLeaf`. (Would it make sense to have methods working on the actors depending on their IDs?)
- At the concrete-model level, `getName()` gives the concrete name associated with any actor (w.r.t. `getActorID()`). For example, `MyComposite`, `MyLeaf`, and my `MyZLeaf`.

12. The method `getPosition()` and `getDimension()`, in the graphic framework, should be made uniform with the method `getLocation()` and `getSize()` (respectively) from AWT.

13. Need a nicer display algorithm for `PLinkingMethod`, where origin and target entities are the same.

14. The method `DPattern.getConstituents()` could be better used. To be removed or to be improved.

15. When an entity possesses an association, the solution displayed throws a `NullPointerException`.

16. The calculus made in the graphic framework (especially in classes `DGraphicalElement` and `DSymbol`) must be rewritten (to simplify and to use only integers, if possible).

17. The current algorithms in `PLink.recognize()` and `PAggregation.recognize()` do not verify the non-existence of the link instance2 $\longmapsto$ instance1 (constraint $\neg$ instance2 $\longmapsto$ instance1).

18. Make a more extensive use of `peNil` in Ptidej Solver.

19. The constraint for the starred `Composite` pattern provides inconsistent results: Solutions with only a `Component` and a `Composite` (no leaves).

20. The list `DPatternRootElement[]` in the class `DPattern` contains instances of `DEntity` and of `DElement` all mixed together. The list should contains only instances of `DEntity`: It makes more sense and simplifies several algorithms. However, this is not such an easy modification. Indeed, mixing instances of `DEntity` and `DElement` helps in displaying first the instances

of `DElement` and then the instances of `DEntity`, creating a nice display. Removing the instances of `DElement` requires to display the instances of `DEntity` (and their instances of `DElement`) and then the instances of `DEntity` alone, to write on top of the displayed instances of `DElement`.

21. Redundant pieces of information are displayed when displaying links or associations.

22. The removal of related solution groups should be based on the components, not the parameters of the transformation? Well, maybe not…

23. Refactor the super-entities reconstruction mechanism in class `SolutionBuilder`.

24. Decouple PTIDEJ and its viewers from PATTERNSBOX JAVA-dependent classes, for later CLAIRE and C++ versions.

25. The inclusion of `java.lang.Object` in the `listOfPEntities` may be an issue when solving the constraints. For example, in the Factory Method pattern, the very presence of `java.lang.Object` generates a set of uninteresting solutions. However, for other patterns, such as Composite (?), the presence of `java.lang.Object` may be required to obtain interesting solutions.

26. Should move the class `patterns.WindowCloser` to the package `patterns.util`.

27. Visibility of names and graphical elements is only set for entities.

28. For some reason, when displaying JHOTDRAW, the classes `LineConnection -<|- PolyLineFigure` and `TextFigure -<|- AttributeFigure` are not displayed as close as possible of their super-classes.

29. Implement the `Fold/Unfold All` menu in the `ConstraintResultsFrame`.

30. Add a progress bar when loading a project. This requires to define a new progress listener and a new progress event for the class `PatternIntrospector`. The class `PatternIntrospector` would then notify its listeners when some progress is made.

31. Implement the `Save image` button to save the display as a BMP or JPEG image on the disk.

32. The exceptions thrown by the instance of `ClassLoader` are too restrictive to load only a subset of an application.

33. In the `ConstraintResultsFrame`, allow the use of the keyboard to go up, down, at the beginning, and at the end of the list.

34. In the Ptidej Solver, add a timing pre-/post-mechanism to record the number of run of a method and the time taken depending on the number of entities and relations.

35. Compute and decrease the constraints complexity.

36. After cloning a subset of the pattern, the super-entities and elements that do not exist or do not point on a entity of the subset of the pattern must be removed. See and clean up method `DEntity.addDElement()`.

37. Must improve the documentation of the classes: Document the classes and interfaces using JavaDoc comments.

38. Must add a `weight` properties in the meta-model, in the class `PatternRootElement` (and the methods required to deal with it).

39. The solution pattern n^{o}14 displayed for the Composite pattern on JHotDraw v5.1 does not show any inheritance links.

40. Rename the packages of the `Ptidej UI Viewer Standalone AWT` project to make them consistent with those of the project `Ptidej UI Viewer Eclipse`.

41. Remove all dependencies on `java.awt` (for exemple, `Point` and `Dimension`) in the `Ptidej`, `Ptidej UI`, ... projects. Introduce graphic-framework independent classes?

42. It seems that the `PatternIntrospector` loading mechanism (using classes `TypeRepository` and `TypeLoader`) works thanks to a lucky error. The directory given to the instance of `ClassLoader` contains the full path to the class files; eg. `C:/.../Original Examples/jtu/tests/composite2/`. The directory should only be the *root* directory; eg. `C:/.../Original Examples/`. The package name gives the rest of the path; eg. `jtu.tests.composite2`. Using the root directory and the package name, it is possible to build the needed full path.

43. While implementing the Eclipse version of Ptidej, I encountered a difficult bug related to `ClassLoader`. When using Ptidej in Eclipse, there are two parallel (and unrelated) instances of `ClassLoader` (three, if we include the system class loader): `Loader [jtu.viewer.eclipse.ui.JTUViewerPlugin_1.0.0]`, from Eclipse; and, `patterns.util.ClassLoader@xxxx`, from PatternsBox. In the method `PatternIntrospector.build()`, the reflection API allows to call the `recognize()` method on the different entities and elements found. The methods `Class.getMethod()` and `Method.invoke()` perform, respectively, the search and the invocation of the `recognize()` method. These methods use the class `PatternIntrospector` and an instance of class `PatternIntrospector` (through `this`). Thus, the instance of `ClassLoader` used for loading the entities and elements, for searching and

invoking the `recognize()` method, and for loading and creating the parameters must be the same (or instances of `NoSuchMethodException` and `IllegalArgumentException` are thrown). Two solutions are possible:

- A unique instance of class `patterns.util.ClassLoader` must be used to load the class `PatternIntrospector`, `Entity`, `Element`... Class `patterns.util.ClassLoader` must handle multiple search directories.
- Two instances of `patterns.util.ClassLoader` are used, one of them being the parent of the other one, for delegation.

44. In the AWT version of PTIDEJ, implement preferences to memorize the window position, the directories, the current source code, the abstract model chosen and the concrete patterns loaded...

45. Refactor the class `jtu.ui.solution.GroupSolution`, does the constructor really need an instance of class `jtu.ui.Canvas`, `jtu.ui.kernel.DPattern`, and the associated visibility?

46. Remove all the unnecessary `!= null` conditions and improve the code performance in the graphic framework (package `jtu.ui.kernel`).

47. The method `PAggregation.recognize()` does not relate the field of type `Vector` with the `add()` and `remove()` methods. Need to use CFPARSE to ensure that the methods `add()` and `remove()` use the field of type `Vector`.

48. The computation of the Composite pattern on JUNIT returns a solution that is a Decorator, not a Composite (i.e., `Test`, `TestDecorator`, `TestSetup`, ...).

49. The building of the solution is not too safe: If I try to load constraint results for the wrong project, PTIDEJ does not complain but throws a not-so-friendly `ArrayIndexOutOfBoundException`. I need to improve the error management for the method `SolutionBuilder.getSolutions(Properties, Pattern)`

50. Clean up the `PropertyChangeEvent` and `VetoableChangeListener` mechanisms. In particular, make consistent the methods of class `Pattern` and `PatternRootElement`.

51. Externalize strings and translate them into French, Spanish...

52. Improve the installer to change the absolute paths of the `.ptidej` project files according the the installation location.

53. Include Hervé Albin-Amiot's new algorithm for container-like aggregation relationship detection.

54. The relationship recognition mechanism should distinguish between *input* and *output* relationships (respectively, parameters and return types) and also should include the number of chained messages (to enforce the Law of Demeter).

55. Memorize in the constraint domain which entities are abstract, concrete, or interface, to check if the root of a class hierarchy are abstract or not.

56. Remove the backward link between instances of classes `PEntity` and `DEntity` in class `jtu.ui.kernel.Pattern`.

57. Remove the link between constituents in the UI framework and constituents in the meta-model.

58. Class `InheritanceClusterLayout` needs some heavy refactoring!

59. The `jtu.solution.test.distorted.Mediator` unit test highlights a difference between the AC4 algorithm (right, the solutions and only the solutions) and the custom algorithm (wrong, too many solutions, the explanations for removing values are erroneous). The custom algorithm must be fixed.

60. Improve the user interface by giving feedback to the user on what is happening (*loading* program architecture, *building* graphic representation, *solving* constraints...)

61. The method `Misc.isPrimtiveType(String)` only works for Java types, it does not work for class file types.

62. For some reason, when exporting Java AWT v1.3.1 as domain with the AC4 constraint domain generator, the generator loops.

63. Distinguish constructors from methods to add more flexibility to the meta-model.

64. The PTIDEJ UI project references the PDL project because it needs to know about the `Pattern` class to build instances of the `GroupSolutionPattern` class. I should remove this reference, that would be more clean.

65. In the `CombinatorialAutomaticSolver`, the dynamically-created problems are always configured to show all variables. They should only show the variable defined in the original problem by the user.

66. The mechanism used to handle constraints of lesser importance should be externalized and made explicit when *posting* the constraints. (It should not be embedded in the constraint definitions.)

67. Claire's garbage collector loops when solving the knowledge distance test problem on a subset of JUnit with the combinatorial automatic solver. This is a serious bug in the GC that prevents the detection of interesting stuff!

68. I should refactor the `PtidejSolverConstraintGenerator` to use method `createConstraint()` everywhere (including for ignorances, inequalities, and inheritances).

69. The `VisitorRepository` should use the `TypeLoader.loadSubtypesFromDir()` method!

70. Link `ConstraintResultsFrame` through the extension mechanism.

71. The `Representation` class needs a better name!

72. I should implement a better layout algorithm, maybe one based on the extended Sugiyama algorithm developped by Holger Eichelberger at the Bayerishe Julius-Maximilians-Universität Würzburg.

73. Improve the integration of Ptidej as a plug-in within Eclipse!

74. When an entity is selected in the class diagram, it should be possible (via a popup or something) to open the corresponds Java file.

75. Implement JUnit tests for the `ExtraInformationProcessor` class!

76. I should clearly distinguish between models (interclass- and design-levels) and patterns. This is important for the sake of explanations in my Ph.D. thesis!

77. I must check whether I really do not need to override the `clone()` method in class `Constituent`.

78. When loading concrete patterns, I should create an instance of the `DesignLevelModel` and add to it the proper entities, pattern models, and elements, instead of having group solution.

79. Since the last round of refactorings (where I introduced the `AbstractModel`, `AbstractLevelModel`, and `PatternModel` classes), some methods are not consistent, for exemple, method `jtu.solution.SolutionGenerator.getSolutions(String, PatternModel, AbstractModel, int, int)` should be `jtu.solution.SolutionGenerator.getSolutions(String, PatternModel, AbstractLevelModel, int, int)`.

80. Improve the feedback mechanism in the Eclipse plug-in when selecting entities or micro-architectures in the Ptidej view and when selecting item in the package view.

81. In Ptidej Solver, the partial order among constraint should be explicit and it should be possible to limit the automated constraint relaxation to a certain type of constraint.

82. When the maintainer puts back a constraint previously relaxed, I should remove the successors of this constraint that have been added!

83. Improve the `PercentLayout` to enable min width and min height (for the button panel of Ptidej).

84. Rename method of `GraphModel` to be consistent.

85. The `metamodel.util.Misc.isArrayOrCollection()` method uses the `Class.forName()` method. It dramatically slows down loading projects.

86. The `metamodel.util.TypeLoader.loadSubtypeFromFile()` method creates (too?) many instances of `com.ibm.toad.cfparse.ClassFile`. This method needs improvements!

87. Rename methods, such as `getDHierarchies()` to be consistent.

88. Improve the `jtu.ui.layout.InheritanceClusterLayout.doLayout()` method to save up memory!

89. Improve the management of property to save up memory!

90. Rename the packages to replace *jtu* (former "Java-To-UML") with *ptidej*, to match the package names in Caffeine.

91. There is a mismatch between the `Visitor` interface, its implementations, and the use that is made of them: Most visitors are called from the UI through the UI-extension mechanism and thus use UI-related information, such as:

    ```
    if (!(targetEntity instanceof Ghost)
        || (this.visibleElements
            & VisibilityElement.GHOST_ENTITIES_DISPLAY)
            == VisibilityElement.GHOST_ENTITIES_DISPLAY) {
    ```

    Maybe there should be two visitors: One to visit models of the meta-model, one to visit model of the graph model. All visitors called from the UI and using UI-related information should implement the visitor on the graph model. These visitors methods would be called only if the related information is shown in the UI, such saving up in complexity!

92. Replace any '/' with `File.separatorChar`.

References

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1^{st} edition, 1994.

[2] Object Technology International, Inc. / IBM. Eclipse platform – A universal tool platform, July 2001.