www.onboard.jetbrains.com

# onBoard

ELECTRONIC MAGAZINE

Issue 2 February 2005

## IN THIS ISSUE

# Listeners Dependency Injection

Mike Aizatsky, JetBrains

*Listeners and events are widely used in modern applications. This article finds a way to explicitly define listener-based component dependencies in generic IoC container. The method leads to simplifying listeners code, separation of event delivery logic, as well as defines listener dependencies in analyzable way. These benefits are used to solve common problems with listeners: the order of event notifications and the problem of exponential growth of events in a system with many components.*

*We've recently started to apply IoC ( Inversion of Control) ideas to our code base in several projects. While it turned out to be quite beneficial I won't advocate for the usage of any container (PicoContainer) or for the IoC idea itself. I will just describe the set of ideas about listeners, which came to my mind while introducing IoC to our code.*

*I would like to get any feed back about these ideas you might have. I also invite you to discuss the article and all listeners-related problems in the accompanying forum.*

*All the code in this article is written without the use of JDK 1.5 generics. Though generics would simplify the idea a bit there're plenty of projects not using the generics yet.*

## The Problem

It's quite common for components in our projects to produce certain kinds of events and to add listeners to each other. For example:

```
public MyComponent
  implements ResourceListener {
  private ResourceManager _manager;
  public MyComponent(ResourceManager mgr) {
     _manager = mgr;
  }

  public void start() {
     _manager.addResourceListener(this);
  }
  public void stop() {
     _manager.removeResourceListener(this);
  }
}

public interface ResourceManager {
  void addResourceListener(ResourceListener l);
  void removeResourceListener(ResourceListener l);
}
```

```
public class ResourceManagerImpl
  implements ResourceManager {
  //all the usual events stuff
  private List myListeners = new ArrayList();

  public void addResourceListener(ResourceListener l)
     { … }
  public void removeResourceListener(ResourceListener l)
     { … }
  private void fireEvent1(Event e) { … }
  private void fireEvent2(Event e) { … }
}
```

So, what's wrong with this code?

1. The code has more dependencies than needed. `MyComponent` doesn't need to know that this kind of events is actually coming out of `ResourceManager`. The situation is even worse. We often have situation when `ResourceManager`'s implementation is quite complex, and it's split into several other components. To let them fire events on behalf of the `ResourceManager`, the `fire*` methods should be made public!

2. Every event source has quite a few lines of code devoted to listeners management and event firing. It's nice to factor it out somehow.

## The Solution. Client part

Let's try to simplify the client code – the MyComponent implementation. Since we don't want to know anything about actual event source, we would prefer to write the code like:

```
public interface ResourceEventSource {
  void addListener(ResourceListener l);
  void removeListener(ResourceListener l);
}

public MyComponent implements ResourceListener {
  public MyComponent(ResourceEventSource eventSource) {
     _eventSource = evtSource;
  }

  public void start() {
     _eventSource.addListener(this);
  }
  public void stop() {
     _eventSource.removeListener(this);
  }
}
```

Now we have created the client, which might be initialized by IoC container. It's not a big deal yet. Let's think a bit more about it. Why should we bother with all that start/stop methods? We'd surely like just to express our intention to listen some kind of events. Let's simply write:

```
  public MyComponent implements ResourceListener {
     public MyComponent() {
     }
  }
```

and let the container spot the fact we're implementing the ResourceListener! How will the container know we'd like to listen for events? Several possible ways can be immediately proposed:

1. Combination of reflection and naming scheme. Let the container analyze all components interfaces and look for `*Listener` interfaces and `*EventSource` parameters.
2. Combination of reflection and explicit registration. There's nothing wrong in having this code (xml configuration) somewhere:

```
container.registerListenerClass(
  ResourceListener.class,
  ResourceEventSource.class);
```

3 Annotation scheme. Can be used by both lucky JDK 1.5 guys and mere mortals with XDoclet:

```
/**
*  @listener class= ResourceListener
*/
```

## The Solution. Event source part

Event sources might be greatly simplified with ease too. We don't want to manage all the listeners and write all the fire methods. Let the container provide us with one event sink and let it manage all the stuff itself!

```
public class ResourceManagerImpl {
  private ResourceListener _eventSink;

  public ResourceManagerImpl(ResourceListener eventSink) {
     _eventSink = eventSink;
  }

  private void foo() {
     …
     //fire the event
     _eventSink.resourceAdded(new ResourceEvent());
     …
  }
}
```

In this code, instead of managing the listeners you've got the `eventSink` – it's the event broadcaster, which serves as a single point for you and broadcasts all events to all listeners. And instead of `fire*` method the corresponding listener method is invoked.

## The container logic

The container itself would create the event proxy mechanism for every listener in the system. The proxy should gather all the events from different event sources and reroute them to appropriate clients. The wiring mechanism shouldn't be difficult to implement in any IoC container. In fact I already have the implementation prototype for PicoContainer, which I'm going to publish soon.

## Benefits

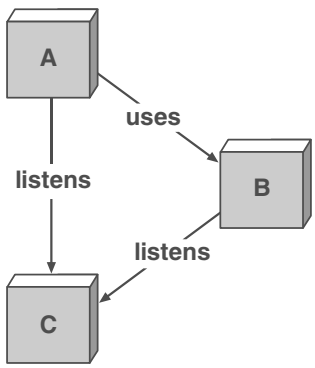
Let's analyze the proposed solution and examine benefits.

1. Event sources are much simpler now.
2. It's possible to have multiple event sources for a single kind of event without exposing this fact to the client and compromising methods visibility.

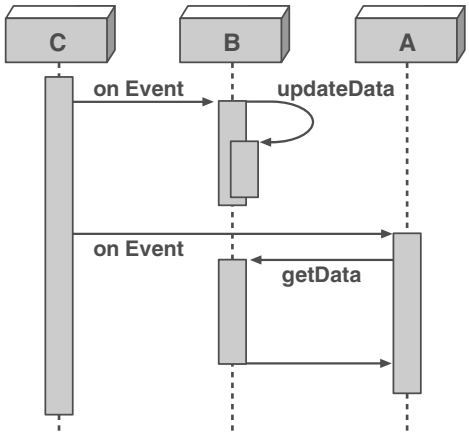
3. Clients are totally isolated from the event sources. They deal with events only. In fact, with this scheme it's even possible to move the event source into a remotely running application without a single change in the client code.
4. Event-based dependencies are stated in an analyzable way both at run & compile time. This fact will be heavily used in the following discussion.
5. Due to the fact that every event is going through a single event broadcaster, the implementation of event distributing algorithm might be changed easily. You can introduce timers, postpone updates, add prioritized queues etc. The order of client notification might be also easily changed.
6. This solution is also well suited for containers that support dynamic component removal/addition – the container itself might dynamically register listeners. Clients no longer need to listen for container events to register/remove listeners when components appear/disappear.

## Another Problem: Listeners and dependent components

The conventional listener mechanisms often fail in the case of interdependent components. Consider the diagram bellow. In many cases this scheme doesn't cause any trouble, until **B** starts to change its own internal state significantly after receiving the event from **C**. The common example of this behavior is caching the data and accessing caches only on later data requests.

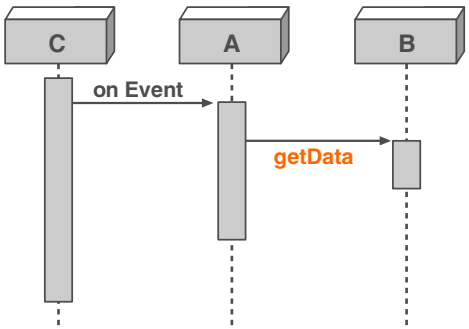



Let's first look at the good processing order:




In this case **B** has received the event first and was able to update its own data. That's why the later request for data from **A** doesn't cause any trouble.

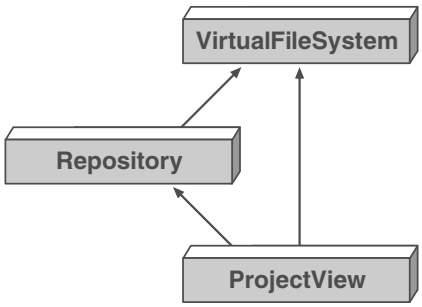
The case with wrong event processing order is straightforward too:




In this case **A** receives the event first and immediately accesses **B** for the piece of information needed for **A** to perform its job. Apparently, this piece of information can't be served because the internal **B** state is not synchronous with **C**'s yet. It will become so only after **B** will process the corresponding **C** event.

Though the use-case might seem a bit artificial, we have plenty of examples of these sorts of problems in IntelliJ IDEA. E.g. we have the `VirtualFileSystem` component, which hides all the differences between file systems on different platforms, jars and even http protocol. We also have some kind of repository, which caches high-level info about java files. It holds the list of classes they contain, methods, etc. The project view displays the information about the project and accesses both the file system, for forming the project tree, and the repository for displaying classes and methods. (The atter request is not done directly, but through the use of other components. I'm simplifying matters a bit.)

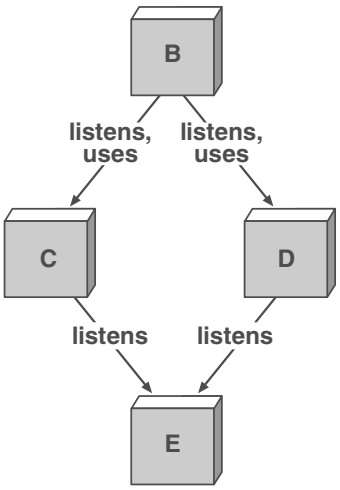
lApparently the project view should listen for events from file system in order to update its UI. The problem fits exactly into the class of problems I've described.




I've seen several ways of dealing with these problems in different projects: introducing priorities for listeners, using `invokeLater` several times before accessing other components in event listener, and even forbidding the access of other components while events are still being propagated. None of them seems to be easy enough to use and maintain.

It's not difficult to see now, how explicit event dependency mechanisms can resolve such problems without introducing overhead into project maintenance. Since we have full, easy-to-analyze information about all kinds of project dependencies we can easily sort listeners and fire events in order, which doesn't cause this trouble. We can even detect the dangerous situation when both **A** and **B** depend on each other.

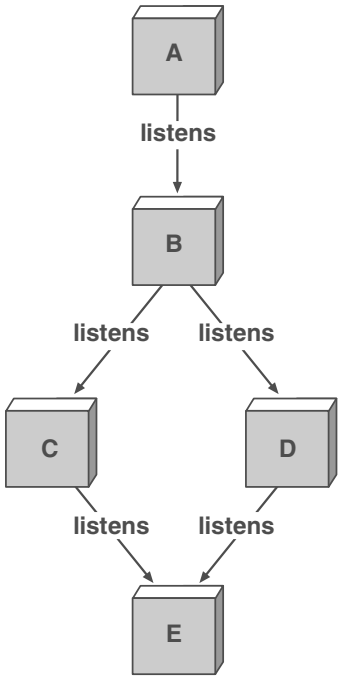
In fact, simply sorting the listeners' order is not really enough for complex component systems. To see why it's true take a look at another dependency diagram.




In this case the event propagation system will probably need to temporarily freeze the event propagation from **C** to **B** while all events from **E** are served, since **B** is dependant from both **C** and **D** and might want to access **D** before **D** has got the event from **E** and form the same erroneous situation we are analyzing.

## Performance Problems

There's another set of problems arising from listeners: the exponential growth of events propagated. Take a look once more at the already familiar dependency diagram:




In cases when a component listens for an event from more than 1 source (**B** in our case), it's quite likely for it to produce several events, corresponding to theone that came from a lower-level component (**E**). While it doesn't seem to be criminal enough to bother about it, it becomes quite a problem when you have hundreds of components. I heard about situations, when thousands of actually duplicated events were generated after a single mouse click.

While there seems to be no fit-into-all-projects solution to this problem, there are still some ideas, how the described mechanism of explicit listeners might help. All of them are trying to identify duplicate events and eliminate them from event propagation cycle.

In some projects it's quite easy to detect duplicate events automatically. To do this, the event propagation system should collect all events **B** fires while processing events from **C** & **D**. After event collection, **B**'s events should be analyzed for duplicates. In simple cases it might be enough to notice, that all events have the same type and the same data. In more complex cases other heuristics, probably applicable to specific project only, should be introduced.

The other way to go is to move all the burden of duplication analysis to event consumers. To do so you need to modify all your listeners to receive the list of events, instead of receiving them one by one. I.e. the listener

```
public interface SomeListener {
  void someEventOccured1(Event1 e);
  void someEventOccured2(Event2 e);
}
```

should become

```
public interface SomeListener {
  void someEventsOccured(AbstractEvent[]events);
}
```

Having listeners in this form makes it possible to collect all the events from **B** and simply deliver them to **A** without writing complex event duplication detection logic in the event processing system. Please note that due to the fact we're declaring all our listener dependencies explicitly, instead of registering all of them in code, it's even possible to migrate some clients gradually to this scheme, since components container might be aware of this listener form.

## Summary

It turned that a simple idea of making component container aware of listener pattern results not only in simpler code. It also solves serious, hard-to-debug problems with listeners in large systems. The proposed solution is not yet implemented by us, but we are actively thinking about adopting it in Fabrique project.

### References:

http://martinfowler.com/articles/injection.html
http://www.c2.com/cgi/wiki?ObserverPattern
http://picocontainer.org/

# TMate – new look at the good old CVS

Alexander Kitaev, TMate Software

***Last moment note on TMate:***

*When this article was already sent to the "onBoard" magazine,* ***TMate 1.5*** *was released and this new release features not only CVS support, but also support for Subversion version control – new and becoming more and more popular VCS. Latest TMate version will unlock Subversion repositories for you as well as CVS ones. You are welcome to TMate Web Site where you may find more details on this new release.*

## Preface - role of version control system in the development.

There was a moment in time when Intel just introduced their 80386 processor that, comparing with the previous 8086 one, was enriched with protected mode, new addressing modes, 32-bit registers and new commands to operate on those longer registers and address space. However, most system programmers (ones who used assembler language) only used new long registers to compute greater numbers and more memory to store more data. It took a lot of time for some of the programmers to understand, that with the new addressing modes they could make their subroutines faster and more effective, and even implement more complex algorithms using less bytes of code.

The same behavior pattern could be seen in the version control system (VCS) usage. It is hard to imagine now that not a very long time ago there were a lot of software development companies that didn't use VCS at all. Instead, there was a shared hard drive or FTP server with application source code, manual conflict solving, and weekly backups. The stone age of software development process is far away now and it is hard to imagine development infrastructure without a VCS. However modern VCSs are frequently perceived as one-way storage systems that resolve conflicts automatically (that is not always true) and make a backup instantly, but in general are still just slightly modified versions of the old code sharing system. Compared with Intel 80386 processor's fate, it is a quantitative, not a qualitative difference between the way people use modern systems and the old time ones. Of course, VCSs themselves have made a huge step forward and continue to progress fast, but we, users of these systems, still underestimate power and value of a VCS in the modern software development process.

Sometimes it is just a question of right tools that help to use VCS smarter. This article introduces TMate – an IntelliJ IDEA plug-in that helps developers to get most of the CVS – the most popular version control system.

## Team Leader's role in software development. How VCS may help

Modern software development companies consist of different people that play different roles in the product development process. There is a distinct definition of the "product" for each role.

A project manager defines product as a feature matrix or feature tree, Microsoft Project Gantt chart and release schedule. A quality assurance engineer defines product as a complex of functional tests and their actual state. A developer's product is a subsystem source code, and assigned open and closed issues in the bug tracking system. The role for which product is mainly a source code is a team leader – the person who serves as a communication channel between developers and project manager, as a translator from the language of source code to the high level language of project management.

As in virtually any area in software development, it is very dangerous to let the high level project state become disconnected from the actual state of the project. That is why the team leader needs exact and up-to-date information about source code state as a whole at any moment, not relying on the daily or weekly developers'

reports. Of course, this information should be easily accessible, and accessible with different levels of detail. It is exactly where VCS should play its role of the important information source, not merely a one-way storage system.

In practice, when using CVS version control system, it is not an easy task to get information out of the CVS. What the user may get is a lot of data which always needs postprocessing to get the source code state and history overview. This inconvenience, plus an understanding of team leader needs, inspired me to write a TMate – a plug-in that allows me as a team leader to be always in touch with the source code, tracking real project development easily.

During development TMate evolved into something more than just a source code tracking tool and as an IDEA plug-in it additionally provides lot of small but nice features that make work with CVS more convenient. Both team leaders and developers will benefit from using TMate – below I will describe the idea of TMate, the most important features and how they help.

## How TMate may help?

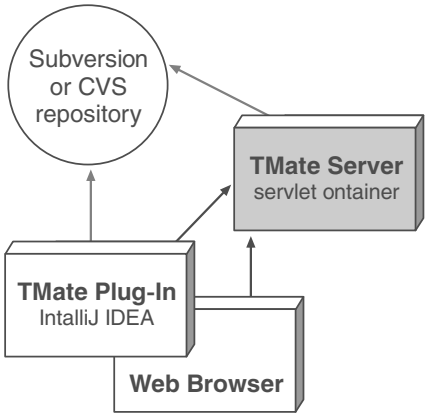
To become popular, software has to be usable. When talking about usability, I mean not only UI slimness, but also (and mainly) the way software behaves. Obviously to be usable, an application should be transparent for the user. It should not require additional information to work, should not intervene into the workflow and at the same time be always close, ready to provide necessary information.

When developing TMate my highest priority was to make it exactly this kind of application, otherwise all the additional value provided by the VCS would not worth the effort of retrieving it. That means that TMate shouldn't require any additional information to provide benefits. This is true both on the large process scale where users do not need to change the way they work with VCS, and on the personal workflow level – the TMate plug-in widely uses modeless feedback, standard IDEA controls and keyboard shortcuts, makes collected information easily accessible.

The name of the product – "TMate" is an acronym for "team mate" – ideal team member that doesn't require any attention but does useful work. Now, when first version of TMate is ready I'm glad to say that TMate justifies its name.

To introduce TMate I will first describe its structure and then demonstrate what kind of information TMate may provide for the certain project, and how.

TMate is a combination of a web application and IntelliJ IDEA plug-in:




The Web application running in an application server (Tomcat 4.0 is included), instantly tracks changes in the CVS repository and represents them as a set of change sets – a concept not supported by CVS, but proved useful in other VCSs. When TMate Server connects to the repository for the first time it indexes all the information that is already there. This may be history for several years of development – TMate indexes it fast, thanks to the smart algorithms implemented in TMate Server.

The TMate web application needs just a read-only access to the CVS repository, which makes it very safe to use and also demonstrates how TMate tries to be transparent, adaptive to the existing environment and thus usable.

The IntelliJ IDEA plug-in contacts TMate Server and displays collected changes. While the web application converts raw data fetched from the CVS repository into meaningful information represented as change sets, TMate plug-in enables you to work with the collected information – browse, search, group, filter, build charts.

## TMate in action

To demonstrate TMate in action, I will take a typical project, configure TMate to track changes in this project and let you see what kind of help you get from TMate. I will use IDEA 4.5.3 and latest version of TMate available in the IDEA plug-in repository.

Installing TMate is very easy, so easy that I won't even describe it in this article. You may find complete installation instructions at TMate documentation page. (Here I may add that it took me around a minute to install TMate with IDEA Plug-in Manager.)

As a typical project that all of you have access to, I will take Jakarta Tomcat that is hosted in the Apache Software Foundation CVS repository. That repository is open for anonymous access and this is enough for TMate. As soon as TMate is installed, it will remind you (with the modeless feedback of course) to configure the TMate web application. In TMate Server configuration, I created a single component named "Tomcat" and filled it with the six modules that I've found in ASF CVS repository.

After applying the new configuration, TMate Server indexed all changes in selected modules and I had to wait no more than three minutes (cool result I suppose, but it depends on a connection speed of course). Of course when TMate server was busy indexing changes nothing prevents me from working in IDEA with an opened project – TMate Server runs in a separate JDK and may run on a different computer as well. TMate server is highly optimized to work fast and with the minimum possible memory footprint – as a good team member that doesn't require others' attention, it requires minimum user and system resources.

So, five minutes after installing TMate, I got a better understanding of what is going on in Jakarta Tomcat project.

The screenshot in Figure 1 demonstrates the main TMate view – an Outlook style tool window with extremely customizable layout. It reuses the time-proven email client mental model, so that there is no need to learn new things. Every modification, like a letter, has date, author and subject - commit message. TMate even marks new modifications as "unread", so that a single glance at TMate window will let you know whether there are new changes or not.

By default TMate displays changes for the last month only, to save memory at runtime and network bandwidth, but it is easy to change or remove this limit in TMate's connection configuration (see Figure 2).

When creating TMate Server configuration, I didn't have a Tomcat project checked out, so now TMate politely suggests me to check out the configured modules. It takes literally two clicks not only to get Tomcat sources from the CVS, but also to configure project modules corresponding to the CVS ones (see Figure 3).

As soon as project modules are created source navigation shortcuts start to work in the TMate tool window. (Just don't forget to enable CVS integration in IDEA project properties.) I'm a keyboard-style user and of course all the famous IDEA navigation shortcuts are supported by TMate. I would even say that there is no action in TMate that couldn't be accessed from the keyboard (see Figure 4).

TMate's view allows changing its layout completely in a few seconds thus allowing taking a look at the source code from different points of view. For instance first you may be interested in a file's history, then in a history of a user's commits. TMate doesn't require any additional information – it is a tool that displays and lets you access existing CVS data in smart way.

There are predefined layouts and a way to quickly create new ones. Needless to say that TMate, like a good assistant, remembers all the changes you make and, tries whenever possible not make you do the same things again and again (see Figures 5, 6).

So now, less then ten minutes after installing TMate, I know with some degree of precision:

- What Tomcat subsystem is under most active development now
- What problems are being worked on
- Which developers are working on them
- What the overall pace of development is

The Tomcat project team uses "Bugzilla" to track bugs and frequently references Bugzilla issues in the commit comments (that is actually a common practice). TMate provides a generic way of integrating source code and bug tracking systems. For the Tomcat project I defined the following regular expression and URL pattern in TMate connections properties:

Regular expression to match issues: "Bugzilla (\d+)"

Pattern to generate issue link:

http://issues.apache.org/bugzilla/show_bug.cgi?id=$1

Now I may navigate to the issues referenced in commit comments with a single click, as shown in Figure 7.

Charts are another representation TMate could provide. Charts give you a certain project aspect overview on a longer-term scale that is not only useful for team leader, but may let team members measure their performance and efficiency in terms of influence on the source code (see Figure 8).

All axes are configurable, as well as stacking value. Charts may be saved to gif or exported to CSV for further processing (i.e. in Excel). All the filters applicable to the changes view may be independently applied to the chart.

## More TMate Features

There are more TMate features that make a developer's life easier.

The Web interface makes TMate information accessible outside of IDEA, useful when one would like to track latest changes only and either does not have the ability, or would not like to start the whole IDE. The Web interface is not as rich as that of the plug-in, but it gives enough information structured in a way far better than the standard CVS web interface.

Developers will find useful the Local Changes tool window and Pending Updates tab in the changes tool window. These are valuable additions to the standard IDEA CVS integration.

The Local Changes window is similar to the IDEA "commit dialog", but allows grouping modified files into change sets and committing them separately. And, what is even more important, it is not modal as "commit dialog". It thus provides convenient navigation from the changes tree to the source code editor and back, and doesn't require the user to go out of the workflow to review changes to be committed. TMate updates the contents of this view automatically, so that it always contains up-to-date information (see Figure 9).

Pending Updates is a view that takes advantage of the constant tracking of the CVS repository and displays project files that were changed by other users and need to be updated. This view allows updating only actually modified files and thus saves a lot of time especially when working with a large project.

This view also highlights possible conflicts (i.e. files modified both in repository and in the local project) and suggests a way to resolve them faster.

I wrote this article on a weekend, and Tomcat project didn't contain any changes I should synchronize my project with. So, screenshot in Figure 10 is taken from the other project.

There are lots of other small, mostly invisible features in TMate that make it a pleasure to use and nice to look at, but I think that instead of reading about them in this article you'll have more fun discovering them yourself.

## Conclusion

TMate of course is not a silver bullet and will not fit into every development process. First of all it works only with CVS version control. CVS in its turn is quite an old tool and a modern VCS, like Subversion, outweighs it with features. But switching to a new VCS is a time consuming and costly process that not every project can afford. These new VCS may not be as stable as CVS. Subversion, for example, lacks good IDEA integration at the moment.

In such situations TMate may be a valuable tool that brings work with the CVS to a higher level without changing anything in the current infrastructure. It is a transparent tool that is able to give bold results.

If you are interested in TMate you may always find more information and get a trial version at TMate Home Page and for those of you who read this article to this very lines and is working on an open source project I have one more good news to share – TMate is free to use in established open source projects – just send me a reference to your project home page and you will get a license for free.

**References:**

http://cvshome.org/ - CVS home

http://subversion.tigris.org/- Subversion home

http://tmatesoft.com/documentation/ - TMate online documentation

http://tmate.org/svn/ - JavaSVN library home page

**Figure 1:**

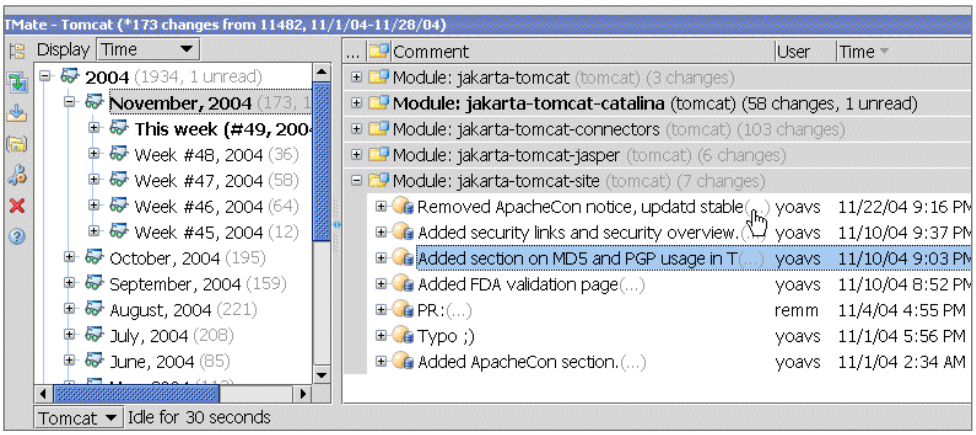



**Figure 2:**

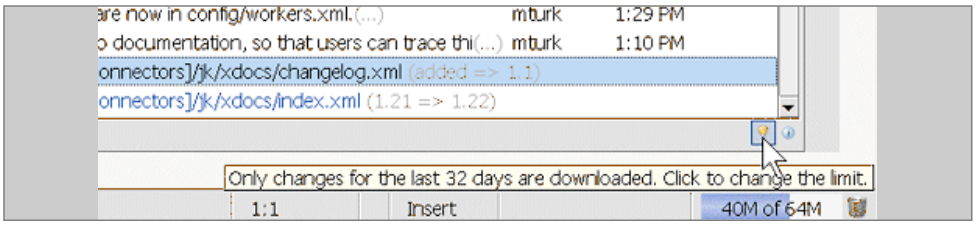



**Figure 3:**

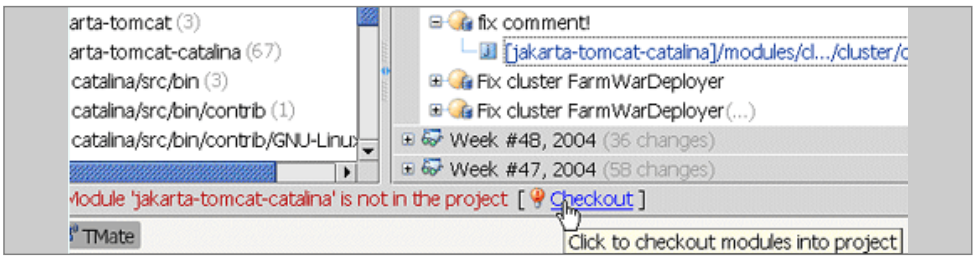



**Figure 4:**

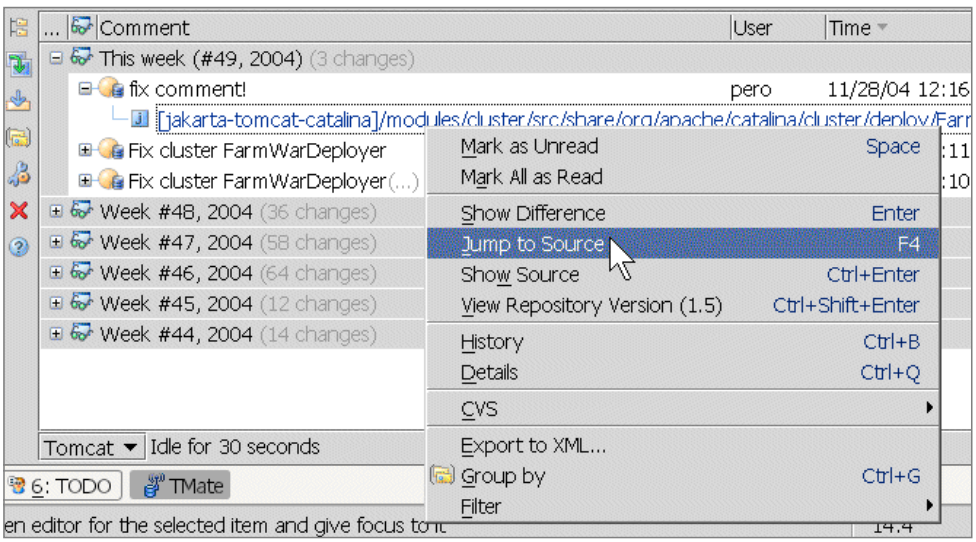


**Figure 5:**

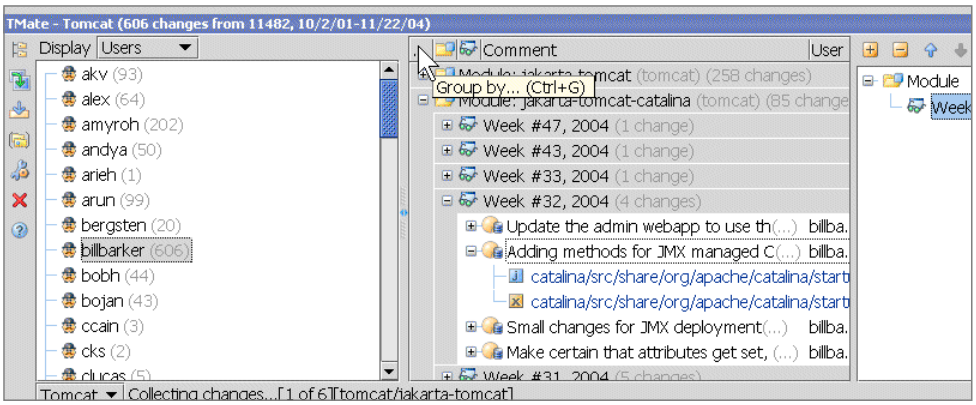



**Figure 6:**

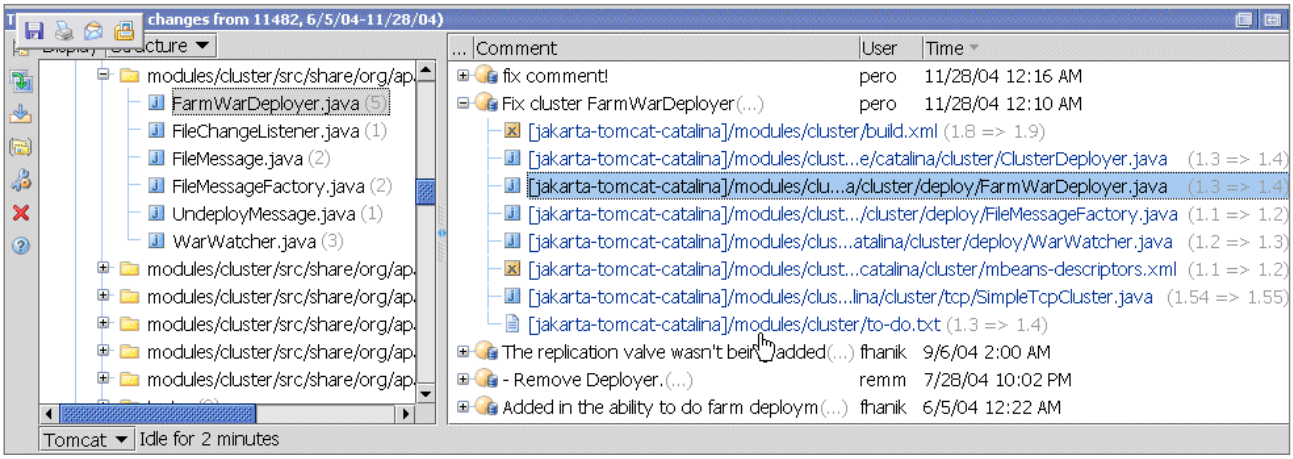



**Figure 7:**

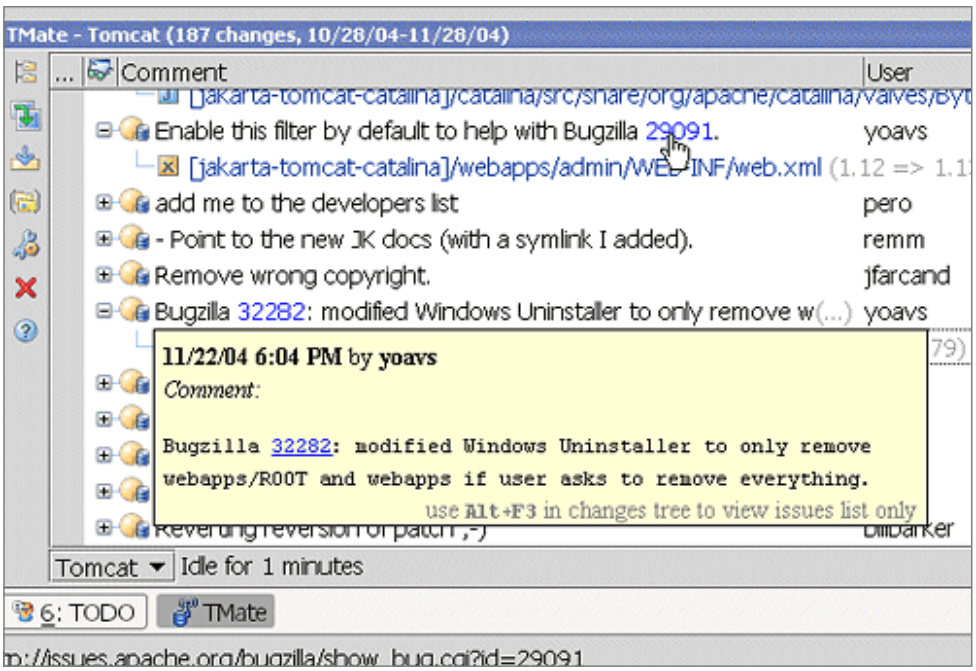


**Figure 8:**

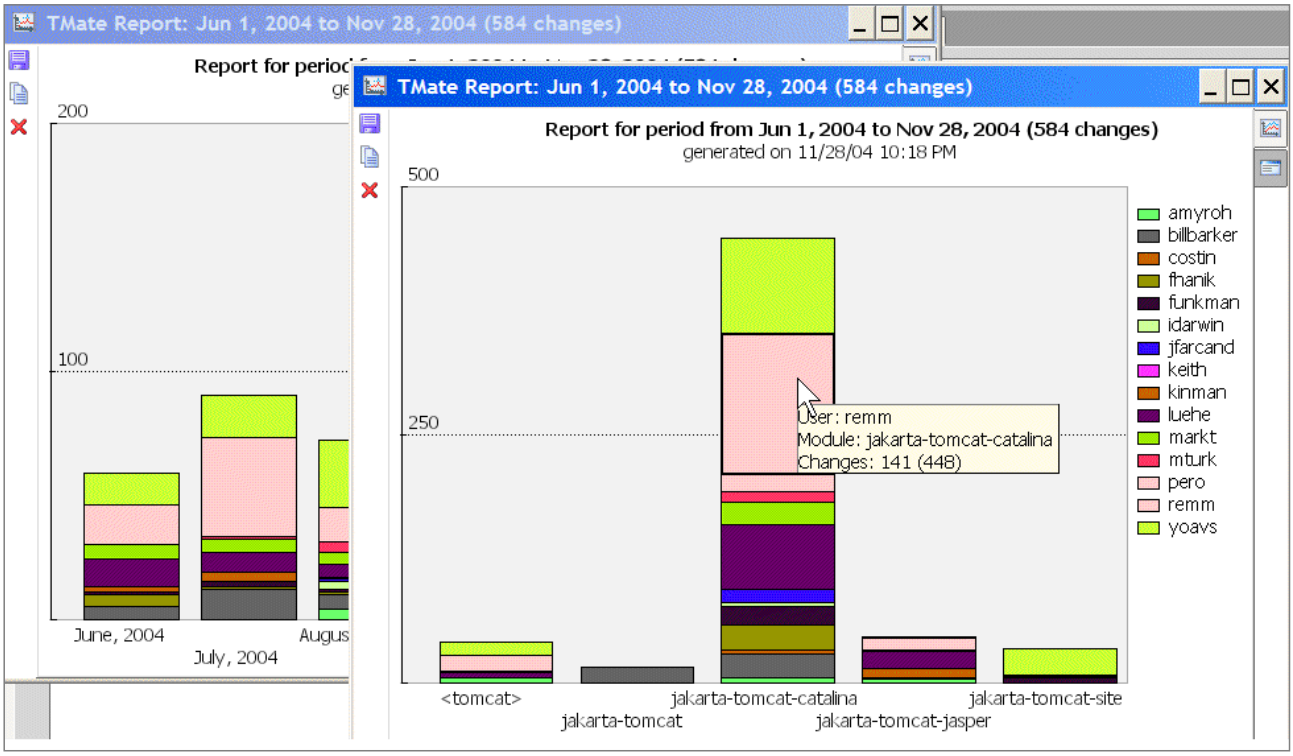



**Figure 9:**

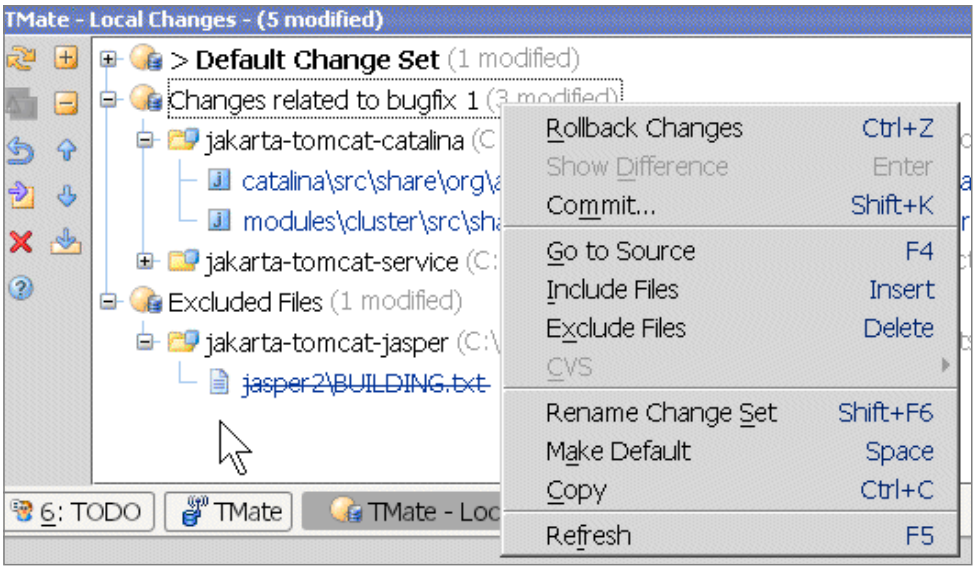
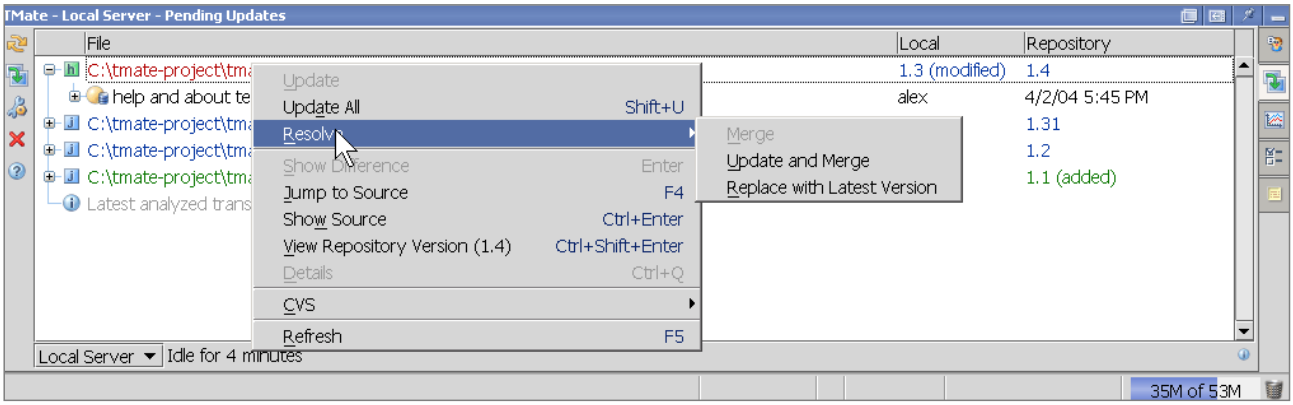


**Figure 10:**



# What's good and what's bad about profiling nowadays

Anton Katilin, YourKit

*There are lots of developers who already understand the necessity of performance tuning of Java applications. They know about profilers, and most likely use them in everyday developer's life. These people reasonably ask: why should we use this or that tool, why one is better than another, and finally which one is the best?*

## Approach to memory profiling

Let's start from the beginning. The first critical issue is memory profiling. Being a developer of a considerably big project, one often faces memory-related performance problems. The program starts "eating" memory, and/or does not reclaim it when it has to. Surely, there are profilers to handle this. But the lion's share of them have one common drawback: first, it takes extremely much time to start a program under the profiler, and then you have to wait long while it reaches the memory leak state. What makes it even worse, the leak may happen only under very rare and even unknown, obscure conditions, and often no one has any idea how to reproduce the problem. And it would be great to be able to analyze the heap of every application once a memory problem is found. Not long ago this was impossible, because one could not always run an application with a profiler - it would have lead to unacceptable performance degradation. That was the case, at least with profilers that existed before. Then I thought: isn't there indeed any way to take all the necessary information from a running Java application when it is needed without making the application slow as hell during its entire run time? After a series of researches and tries, my colleague and I managed to find the way out.

The clue is not to rely on time-consuming processing of events such as object creation or destruction by the garbage collector, i.e. not to record object allocations. To find and fix a memory leak, it is not that interesting to know who created the object, but rather who holds it, so that this approach works perfectly. Instead, there can be an **optional** ability provided, to turn object allocation on/off when needed, for those who need [I'd better say 'want' or 'think they need'] it. Frankly speaking, I personally do not find it useful to know where a particular live object is created. The useful aspect of allocation recording is to find the code that produces a lot of 'garbage', i.e. temporary objects. Modern garbage collectors work fast and temporary objects are not that big a problem. Anyway this optimization matters, because it always takes less time [for a garbage collector] not to do a job at all, rather than do it even if it is done fast.

The deep analysis of the memory profiling problem turned into effective result: we have developed such a tool (namely YourKit Java Profiler), with which an application can run with NO overhead, and is ready to tell all about its memory state exactly WHEN it is needed. Currently YourKit is the only profiler that addresses the issue of memory profiling adequately and in a user-friendly way.

## Approach to CPU profiling

An approach similar to that discussed in the previous section can be applied to CPU profiling. An application starts and runs at full speed. When the need arises, the profiler is activated; it starts recording profiling data, which of course may lead to a performance overhead. This lasts until the application finishes the particular task you are interested in, and then you capture a snapshot with all collected information. The application goes its way, again at its full speed. You go your way - open the collected information in the profiler UI and study it.

The purpose of the tool matters. A useful profiler should not be just a tool primarily targeted to show beautiful views jumping synchronously with an application being analyzed, with no regard to how much this "live show" slows down the application. Alternatively, it should help finding slow parts of a running application and show or give a hint how to make them faster. This theory became the basis for YourKit profiling ideology.

This may be easily taken as radicalism. And radicalism is not always good. Thus we are currently working on adding support for some useful "telemetry" information that illustrates on application's time line. That should be an addition to the main "start - wait - capture snapshot" approach.

## Filters

In general, measuring is intrusive. That's a law of nature. No measuring tool can measure without affecting the thing it measures, either in the software world or in the real physical world. Capturing a memory snapshot may pause the profiled application for a couple of seconds, CPU sampling gives a very small, invisible, non-perceptible but still existing constant overhead for periodic inquiring into stacks of running threads. When the CPU is profiled with tracing (which handles method entries and exits), or an option to record object allocations is used, intrusion is more significant.

There's an approach for reducing this intrusion by means of reducing the scope of code (methods, classes) to measure (profile). As a result, the code you measure runs slower, but the rest of the code runs fast. So you can skip profiling of non-interesting parts of an application, and profile only interesting parts. This approach is widely used in different profilers.

It's all about saving *your* time. But... the saving is doubtful.

This approach works, but there's a problem with it. Which appeared earlier: the chicken or the egg? I should be interested in the profiling of slow parts of my application, the particular code that causes performance problems. But how can I know which part of my code is "problematic" and which is not, i.e. which classes, entry points I should profile, other than learning this from profiling results? Of course sometimes it is a priori known, but sometimes not, because I cannot hold my entire project details in my head. Even libraries, the good candidates for skipping "by default", should sometimes not be skipped. For example, in my own practice there were cases when without analyzing the behavior of core Java library classes (thanks to Sun the sources are available) it was very hard to understand the reason for a performance problem.

As a workaround, this approach can be applied iteratively: launch profiling session, analyze results, reduce scope of profiled code, launch another session, etc. But:

- This takes human time and human attention, which is more valuable than computer time. Were's **my time** saving?
- What if I understand the need to change profiled code scope, but cannot re-launch the profiling session? E.g. there's a problem which is very hard to (or unknown how to) reproduce.

In YourKit Java Profiler, we decided to use the alternative approach: in two words, no filtering in runtime. And I bet it does save time:

- Profiling scope is reduced by reducing time periods of heavily intrusive profiling:
  - in most cases, heavily intrusive profiling modes, such as allocation recording and CPU tracing, are not used at all; allocation recording is optional for memory profiling; sampling as a CPU profiling method gives good results in most cases, having very small overhead
  - one can turn on the heavily intrusive profiling modes when needed only, and for limited periods of time
- Human needs to think less, and that's good. Snapshots contain all the data, and filters are only a UI option. You always can change your mind without re-launching measuring sessions. A human has the right to make a mistake and to change his mind, and that's good.

## IDE integration

Tools should be integrated; otherwise it's not easy to work with them. Developer's tools should be integrated into developer's IDE.

What kind of tasks can the profiler's IDE integration provide? Basically, there are two:

- launching profiled program from the IDE with minimal, or better yet, no inconvenience
- providing easy access to the source code of the profiled application from profiling results

**1.** It is logical to seamlessly incorporate profiler-specific commands and settings into the native IDE user interface:

**a)** The *Profile* action is convenient to add to the IDE menu, so that it can be called like any other command. For example, in IntelliJ IDEA YourKit adds the *Profile* action where similar *Run* and *Debug* actions are allocated (namely under the main **Run** menu, in context menus, on the toolbar), as shown in Figure 1.

**b)** Additional profiler-specific settings should be incorporated into the IDE Run Settings, as it is, e.g. done in the IntelliJ IDEA integration (see Figure 2).

**c)** The output console should not be different from the ones provided by the IDE. It should look familiar, in order not to produce any confusion. Taking IntelliJ IDEA integration as an example, we can see that the profiler console is absolutely the same as that used by Run and Debug commands, as it looks and behaves the same way (see Figure 3).

**2.** Many profilers still have code access problems, as they can open the source code only in their own editors, which is not convenient at all. The first thing that crosses one's mind when talking about quality integration, is the ability to view the source code (coming from profiling results) exactly where one usually works with code, i.e. in the native IDE editor. At the same time, there should be some dedicated standalone UI for analyzing profiling information, specifically designed for convenient output. The last two statements may sound mutually exclusive, but when this theory is turned into reality, it becomes simply the right tool (see example in Figure 4).

So, these were the two major issues. And there are obviously more requirements for a good profiling tool.

Having summarized the above, I would say that the following main features define an integrated profiler that covers the most important developer needs:

- quick and transparent setup process
- support for the most popular IDEs (e.g. IntelliJ IDEA, Eclipse, or Borland JBuilder)
- use of IDE-specific UI, possibly along with own standalone UI
- one-click ability (with devoted shortcut) to launch the profiled application from an IDE
- use of the native IDE editor for showing the source code from profiling results

The only question that might still remain concerns the use of the standalone UI. This is, in some sense, a matter of preferences, and many developers would prefer inlined views. This is the feature we are already working on, to provide this option for the YourKit users who do not want to leave their IDE at all.

## Usability

Feature sets of same-purposed products are often similar. But products are not equally easy and comfortable to use. Details make the difference.

The usability aspect in a profiling tool is worth mentioning. There are several main usability drawbacks that are often met in different profilers:

- Many existing profilers are mouse-centric, while keyboard is the main working device of a developer. Vendors of big IDEs, having analyzed why many developers still prefer text editors, have started applying the keyboard-centric approach in their products. The same is topical for other everyday tools that developers work with, and profilers are no exception.
- Working with regular profilers, a user is forced to once again configure the existing project parameters, like directories for compiled classes, sources, or jars used. Is it really a good idea to make a profiler a kind of "launching center" for the profiled application? It seems much easier to add a very small profiler-enabling stuff to the existing configuration, rather than make a user repeat all project configuration steps again, just for the profiler. This can be achieved by just adding one parameter to the application/project.
- An inconvenience is also met in profilers that open the "problem code" in their own windows, while it is obvious that the ability to work on code in the preferred environment is what a developer would prefer. Opening the "native" IDE editor would provide access to all the advanced IDE features, which would make the fixing process really user-friendly.
- When it comes to manual configuration for profiling an application (for whatever reason), specifying correctly all the necessary parameters takes much time and is not always an obvious procedure. For example, setting `bootclasspath`, to enable profiling in an application directly, can sometimes be a rather painful and error-prone task. Why not making things easier, like by just adding the application command line parameter that would point to the profiler agent library?

Having analyzed the existing drawbacks and possible ways to get rid of them, we realized that applying these solutions would definitely brighten the life of those developers who value usability and comfort in their daily work.

All of these usability improvements, as well as many others, are already implemented in YourKit. And there are more to come.

**Figure 1:**

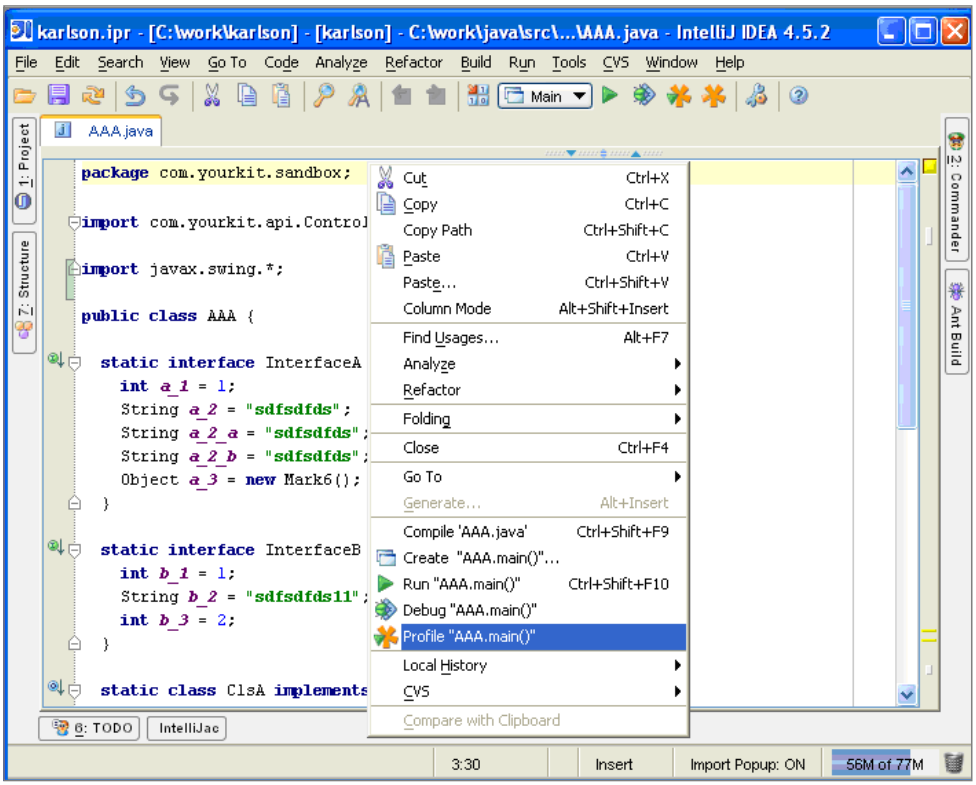



**Figure 2:**

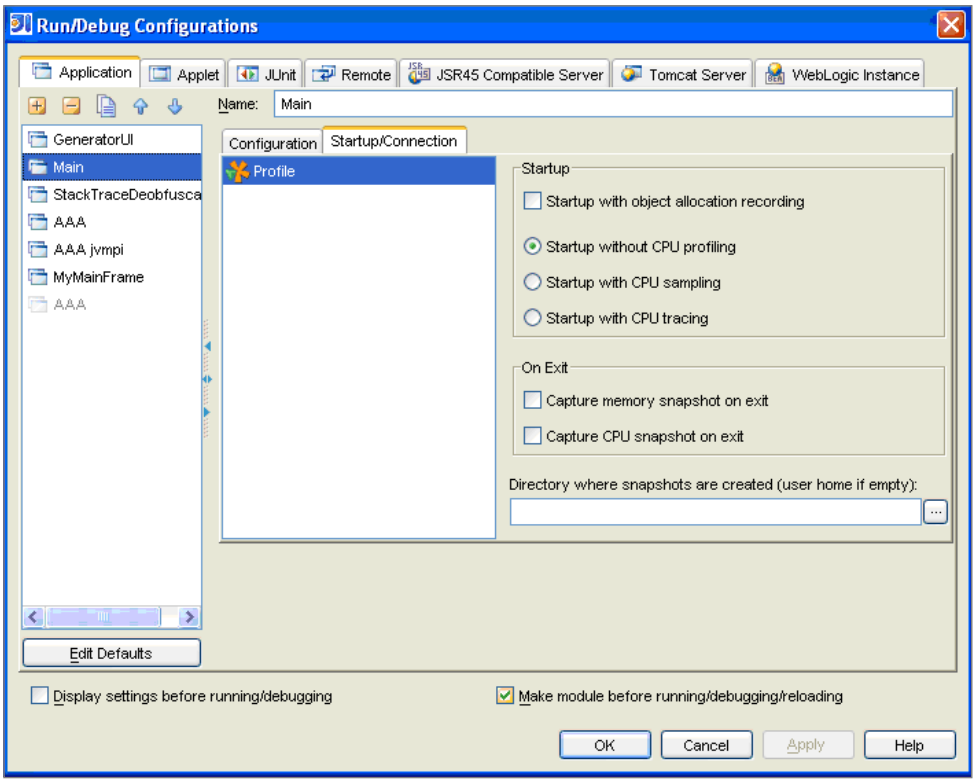


**Figure 3:**

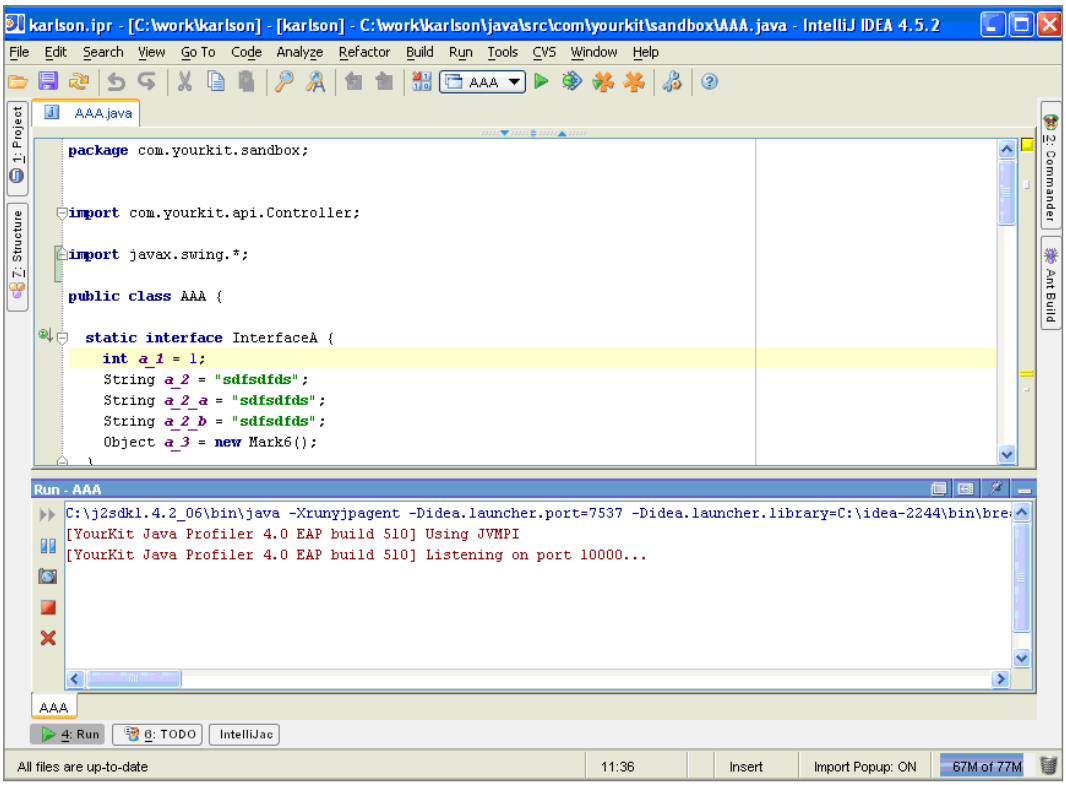
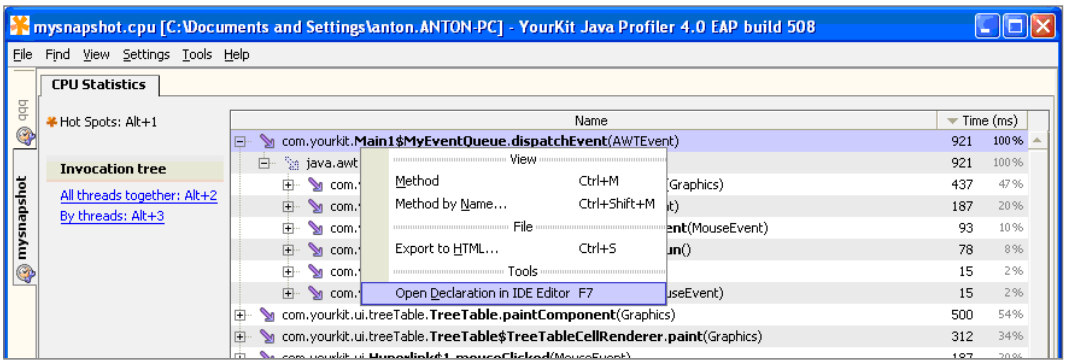



**Figure 4:**




## References:

http://www.yourkit.comYourKit Java Profiler website

http://forums.yourkit.com YourKit users forums

can release a final feature list. However, if you don't want to wait for it, and in fact, want to help shape the feature list yourself, you can do so by participating in its Early Access Program (http://www.jetbrains.net/confluence/display/NetProf/Home).

*David Stennett / .NET Sales Executive / JetBrains*