Inline Visualization of Concerns

Nalin Saigal and Jay Ligatti
Department of Computer Science and Engineering
University of South Florida
{*nsaigal, ligatti*}*@cse.usf.edu*

***Abstract*—Code modularization provides benefits throughout the software life cycle; however, the presence of crosscutting concerns (CCCs) in software hinders its complete modularization. This paper describes IVCon, a tool with a novel approach for completely modularizing CCCs. IVCon enables users to create, examine, and modify their code in two different views: the *woven view* and the *unwoven view*. The woven view displays program code in colors that indicate which CCCs various code segments implement. The unwoven view displays code in two panels, one showing the core of the program and the other showing all the code implementing each concern in an isolated module. IVCon aims to provide an easy-to-use interface for conveniently creating, examining, and modifying code in, and translating between, the woven and unwoven views.**

***Keywords*-code modularization; crosscutting concerns; refactoring**

I. Introduction

Code modularization provides many software-engineering benefits; it makes code easier to write, understand, and maintain. Conventionally, software engineers try to separate code segments that are orthogonal in their functionality into distinct modules. However, in practice, software does not decompose neatly into modules with distinct, orthogonal functionality. For example, code that displays a popup window notifying users about a failed login attempt may be present in a login module, while (partially) implementing various other functional concerns such as security, GUI, and authentication; it may be equally reasonable for the window-popup code to be located in a security, GUI, or authentication module, and at various times it may be more convenient to write, view, or edit the window-popup code in the context of these other modules. Tarr et al. call this problem the "tyranny of the dominant decomposition" [1]. Although it is useful to modularize the same code segment in various ways throughout the software life cycle, current programming paradigms only allow modularization in fixed and limited ways (e.g., into functions and objects).

Conversely, functional concerns of software typically require many lines of code to implement, and that code is usually *scattered* throughout several modules. For example, code implementing a security concern may be scattered throughout login, logout, and network-socket modules. Thus, code segments implementing a functional concern may crosscut through other functional concerns of the program; such code segments implement *crosscutting concerns* (CCCs). Modularizing a CCC involves collecting and displaying in one place all the scattered code implementing that CCC. Isolating concern code in this way benefits programmers because it relieves them from having to browse through the whole program to find, study, or update a single software concern. For example, updating security code scattered throughout an application is much more difficult than updating an isolated security module.

This paper describes IVCon, a tool that provides a novel approach to modularization of CCCs. IVCon users can switch back and forth between two equivalent views of their code, the *woven view* and the *unwoven view*. The woven view displays program code in colors that indicate which concerns various code segments implement (based on users' explicit assignments of those code segments to concerns). The unwoven view displays code in two panels, one showing the *core* of the program (i.e., all code not assigned to any concern) and the other showing all the modularized concerns, each displayed in isolation. Users can create, examine, and modify code in both views. The process of extracting concerns from the main code to produce the unwoven view is called *unweaving*, whereas the process of inlining concerns into the core program to produce the woven view is called *weaving*. Although we have implemented, and this paper discusses, a prototype version of IVCon in a Java environment, we believe the principles underlying IVCon apply to software in any language.

IVCon permits many-to-many relationships between concerns and code. That is, users can assign scattered code segments to the same concern and can assign a single code segment to multiple concerns. We call code that has been assigned to multiple concerns *multi-concern code*.

In addition, IVCon enforces *token-level granularity* in concern assignment; code assigned to a concern must begin at the beginning of a source-language token and end at the end of a source-language token. Allowing finer granularity in concern assignment (e.g., character-level granularity) would be inappropriate because tokens are the core semantic units of programming languages and of concerns implemented in those languages. On the other hand, requiring coarser granularity in concern assignment (e.g., line-level granularity) would be inappropriate as well. For example, consider the following code:

Tool	Can edit code in dual views	Isolates concerns	Can modify identical concern code in one place	Allows many-to-many relationships between concerns and code	Provides GUI	Granularity of concern assignment
Aspect-jEdit [2]	√	√	-	-	√	Line-level
Visualizer [3]	-	-	-	-	√	Line-level
AspectBrowser [4]	-	-	-	√	√	Character-level
SoQueT [5]	-	-	-	√	√	Character-level
FEAT [6]	-	-	-	√	√	Character-level
CIDE [7], [8]	-	-	-	√	√	Nodes in ASTs
Hyper/J [9]	-	√	-	√	-	Declaration-level
C4 [10]	√	√	-	-	-	Line-level
IVCon	√	√	√	√	√	Token-level

Figure 1. Comparison of concern-manipulation tools

```
    JOptionPane.showMessageDialog(mainWindow.frame,
          "Welcome " + userName +". Current time is " +
          format(System.currentTimeMillis()), "Welcome",
          JOptionPane.INFORMATION MESSAGE );
```

Token-level granularity enables assignment of just the `System.currentTimeMillis()` code segment to a `SystemCall` concern, while coarser concern-assignment granularities, such as line- or statement-level granularity, lack the precision needed for such a concern assignment. With token-level granularity, a user could even assign just the method name `currentTimeMillis` to the `SystemCall` concern. At the same time, token-level granularity prevents unreasonable concern assignments possible with finer (e.g., character-level) granularities, such as assigning just the '`i`' in `JOptionPane` to its own concern; this would be unreasonable because if '`i`' implements a concern C, then the rest of the `JOptionPane` token must implement C as well (the `JOptionPane` token has a semantics in the programming language, while the '`i`' alone does not). We consider token-level granularity to be the perfect balance for assigning code to concerns.

Concern assignments in IVCon can also be thought of as labels that document the functionality, or other grouping, of code segments. In this sense, IVCon serves as a code-documentation tool, with the benefit that software engineers can view and edit, in one module, all the code documented as being relevant to any concern.

A. Related Work and Contributions

As a concern-visualization and -management tool, IVCon contributes features beyond those of existing tools, which are summarized in Figure 1. Most of the existing research on concern management restricts users to defining one-to-many relationships between concerns and code, while other tools do allow many-to-many relationships but have line-, statement-, or character-level granularity in concern assignment and lack IVCon's support for conveniently modularizing and isolating concern code. Hence, this paper contributes a tool design and implementation that does all of the following:

- Enables programmers to conveniently translate between, and edit code in, woven and unwoven code views
- Isolates concern code in the unwoven view
- Enables users to modify all identical concern code in one place in the unwoven view
- Allows many-to-many relationships between concerns and code
- Provides a GUI for visualizing and modularizing concerns in software
- Enforces token-level granularity in concern assignment

By providing all these features, IVCon users can flexibly create, examine, and update code in, and translate between, woven and unwoven views of software.

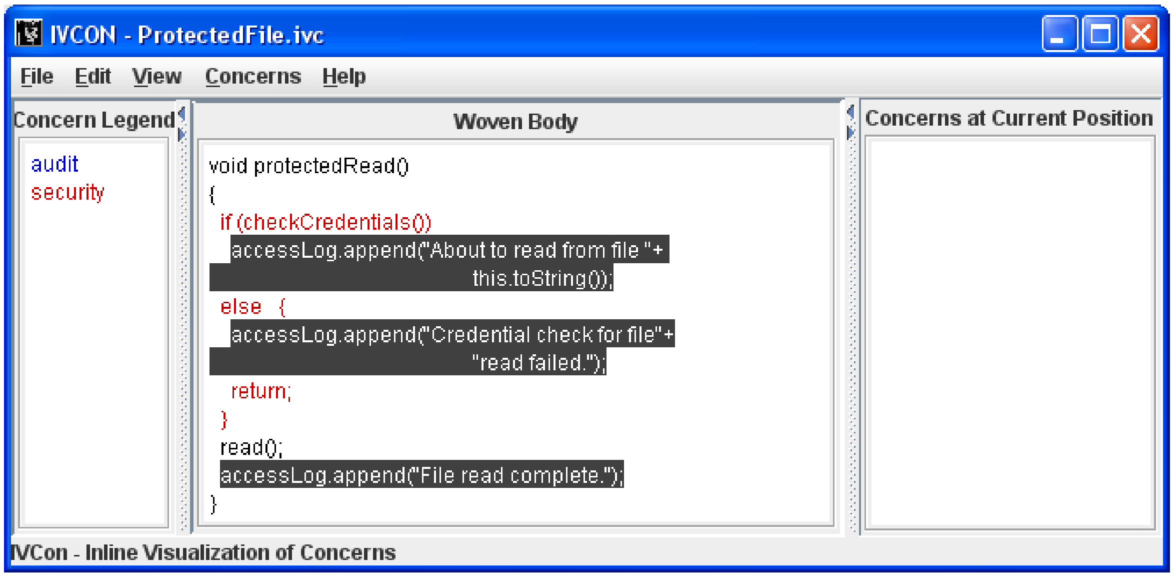
Roadmap: The rest of this paper is organized as follows. Section II describes IVCon's GUI, Section III discusses our prototype implementation, Section IV briefly evaluates the performance of our implementation, and Section V concludes.

II. User Interface

IVCon displays code in two different but equivalent forms: the woven view (Figure 2) and the unwoven view (Figure 3). Users can translate their code between the two views simply by selecting the `weave` or `unweave` menu options, or by pressing <ctrl-w> or <ctrl-u>.

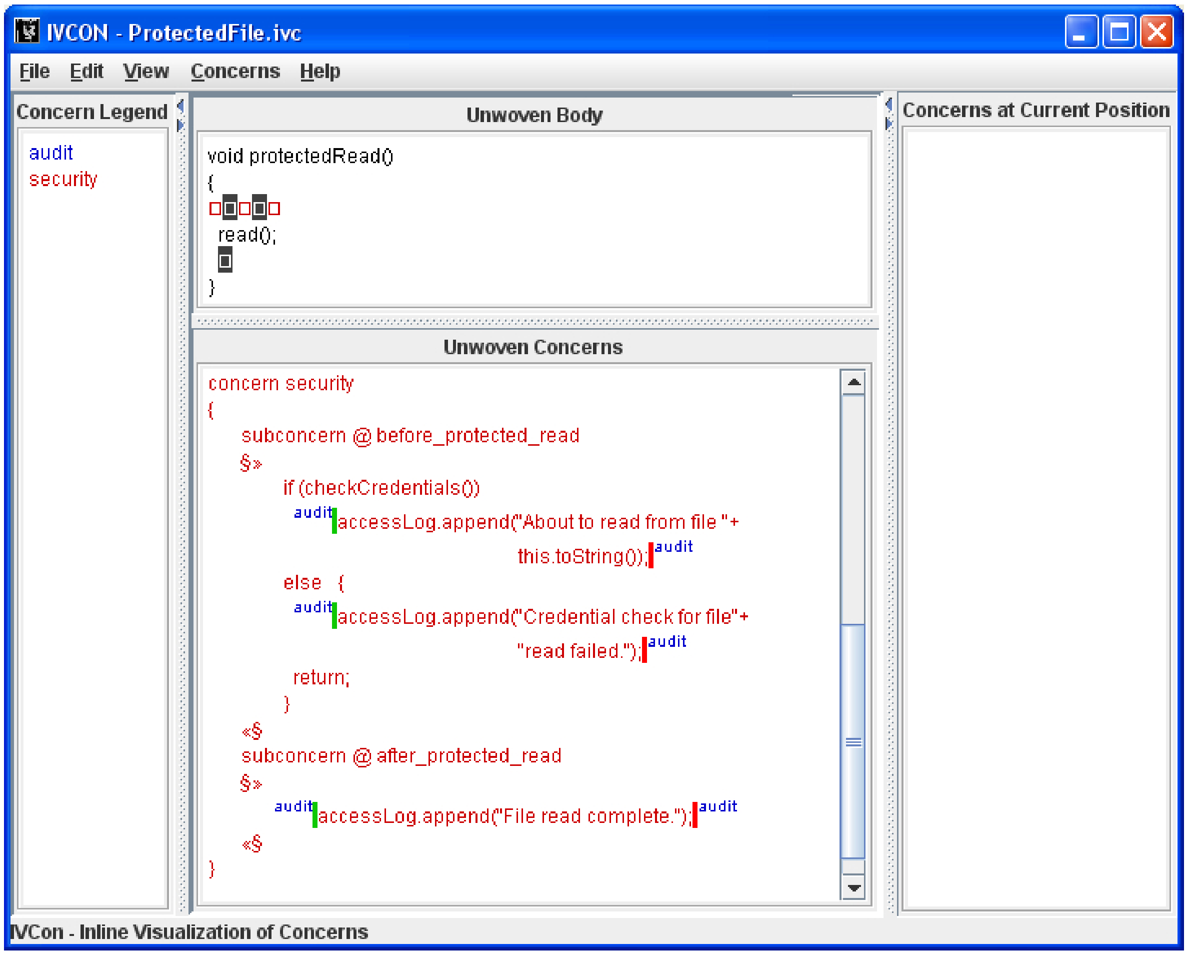
A. Woven View

In the woven view, shown in Figure 2, users can write code as they normally would in a standard text editor or development environment. In addition, users can define concerns, associate a color with each concern, and highlight and explicitly assign code segments to concerns (by right-clicking highlighted code and selecting the concern to which to assign it). Upon assigning a code segment to a concern,


IVCON - ProtectedFile.ivc
File Edit View Concerns Help
Concern Legend
audit
security
Woven Body
Concerns at Current Position
IVCon - Inline Visualization of Concerns

```
void protectedRead()
{
 if (checkCredentials())
   accessLog.append("About to read from file "+
                          this.toString());
 else  {
   accessLog.append("Credential check for file"+
                          "read failed.");
   return;
 }
 read();
 accessLog.append("File read complete.");
}
```

Figure 2. IVCon's woven view of the same code shown in Figure 3


IVCON - ProtectedFile.ivc
File Edit View Concerns Help
Concern Legend
audit
security
Unwoven Body
Unwoven Concerns
Concerns at Current Position
IVCon - Inline Visualization of Concerns

```
void protectedRead()
{
□■□■□
 read();
 ■
}
```

```
concern security
{
   subconcern @ before_protected_read
   §»
       if (checkCredentials())
        auditaccessLog.append("About to read from file "+
                          this.toString());audit
       else  {
        auditaccessLog.append("Credential check for file"+
                          "read failed.");audit
        return;
       }
   «§
   subconcern @ after_protected_read
   §»
      auditaccessLog.append("File read complete.");audit
   «§
}
```

Figure 3. IVCon's unwoven view of the same code shown in Figure 2

IVCon recolors the assigned code to match the concern it implements. IVCon displays code assigned to multiple concerns in white text on a dark background, though users are free to change this *multi-concern background* color. Users can still edit all code after defining concerns; newly typed code gets assigned to the concern(s) present at the position where the user started typing.

By highlighting a contiguous code segment and assigning it to a concern, a user defines a *region* in the code. Every region starts at the beginning of code assigned to a concern and extends as far as the code does that implements that concern. Multiple concerns can share the same region if code implementing those concerns begins and ends at exactly the same positions in the program file. Although IVCon requires a user-specified name for every unique region a user defines, it always provides a default name for regions (based on the name of the concern to which the region is being assigned). If a previously defined region gets assigned to a new concern, IVCon simply reuses the existing region name. Specifying names for regions helps users understand where subconcerns are implemented, as discussed in Section II-B.

Figure 2 shows the woven-view window divided into three panels:

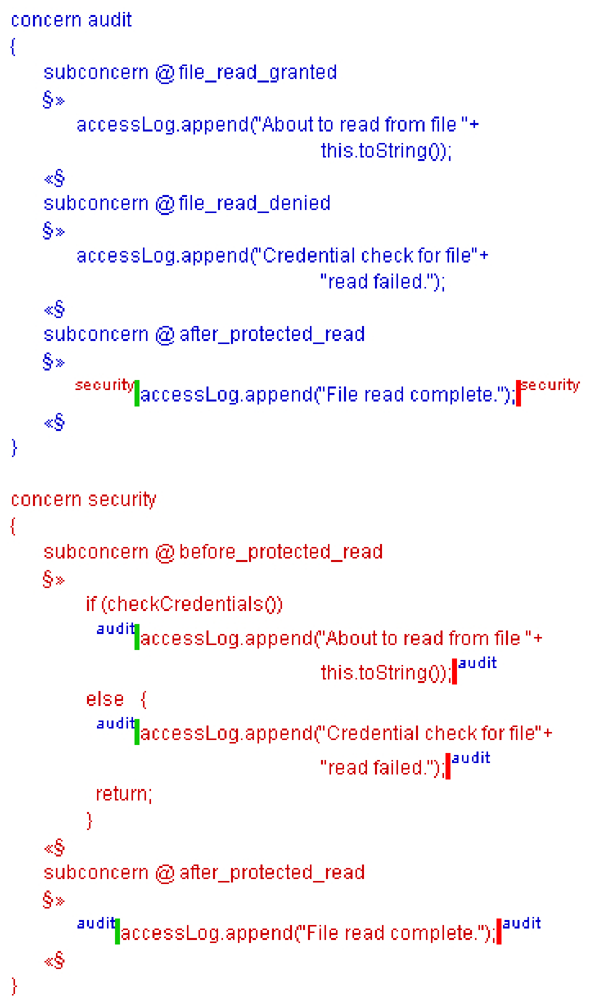
- The concerns-legend panel lists all the user-defined concerns in the current file. IVCon displays the name of each concern in the color associated with that concern.
- The woven-body panel contains user code displayed in colors that indicate concern assignments.
- The concerns-at-current-position panel lists the concern(s) implemented by the code at the current cursor position.

Apart from creating and modifying code, defining concerns, and assigning code to concerns, users can edit concern names and colors, remove concerns, de-assign code from concerns, change the multi-concern background, rename regions, and open, save, and close files in the woven view.

B. Unwoven View

The unwoven view, shown in Figure 3, displays the program core and each concern in isolation. The unwoven-view window is the same as the woven-view window, except that the unwoven view divides the woven-body panel into two subpanels:

- The unwoven-body panel displays the core of the program. Code that has been assigned to one or more concerns is extracted (into the unwoven-concerns panel) and replaced by holes (☐) of the same color as the extracted code. Thus, holes indicate extracted concern code.
- The unwoven-concerns panel displays in isolation each of the program's concerns (as extracted from the unwoven body). Each concern is divided into *subconcerns*, which are syntactically different code segments assigned to the same concern. IVCon displays subconcerns in two parts: a list of the regions in which the subconcern appears and then the subconcern code itself. For example, the unwoven-concerns panel in Figure 3 displays the `security` and `audit` concerns in the format shown in Figure 4. On clicking any region name in the unwoven-concerns panel, IVCon automatically focuses the unwoven-body panel to show that region's location in the context of the program core.



```
concern audit
{
    subconcern @ file_read_granted
    §»
        accessLog.append("About to read from file "+
                                this.toString());
    «§
    subconcern @ file_read_denied
    §»
        accessLog.append("Credential check for file"+
                                "read failed.");
    «§
    subconcern @ after_protected_read
    §»
        security accessLog.append("File read complete."); security
    «§
}

concern security
{
    subconcern @ before_protected_read
    §»
        if (checkCredentials())
          audit accessLog.append("About to read from file "+
                                this.toString()); audit
        else  {
          audit accessLog.append("Credential check for file"+
                                "read failed."); audit
          return;
        }
    «§
    subconcern @ after_protected_read
    §»
        audit accessLog.append("File read complete."); audit
    «§
}
```

Figure 4. Concern-module formatting in IVCon

Code for the `security` concern in Figure 4 indicates the presence of two subconcerns (at regions `before_protected_read` and `after_protected_read`). The code segments beginning with `if (checkCredentials())` and `accessLog.append` implement those subconcerns. Similarly, code for the `audit` concern indicates the presence of three subconcerns (at regions `file_read_granted`, `file_read_denied`, and `after_protected_read`). The unwoven-concerns panel

Unwoven Body:

```
if (buffer.getSize() > □)
   buffer.truncate(□);
if (getTimeElapsed() > □)
   JOptionPane.showMessageDialog(frame,
           "Request timed out","Error",
           JOptionPane.ERROR_MESSAGE);
```

Unwoven Concerns:

```
concern constant
{
  subconcern @ max_buffer_size_0
             @ max_buffer_size_1
  §≫
    512
  ≪§
  subconcern @ timeout_ms_0
  §≫
    2000
  ≪§
}
```

Figure 5. Unwoven view of the code in Figure 6

Woven Body:

```
if (buffer.getSize() > 512)
   buffer.truncate(512);
if (getTimeElapsed() > 2000)
   JOptionPane.showMessageDialog(frame,
           "Request timed out","Error",
           JOptionPane.ERROR_MESSAGE);
```

Figure 6. Woven view of the code in Figure 5

may also contain constructs called *flags* (e.g., $^{\text{security}}$🟩 and 🟥$^{\text{security}}$), which convey information about concern assignment in multi-concern code segments. Section II-C provides additional explanation of IVCon flags.

Figure 4 also demonstrates the usefulness of having descriptive, user-specified names for regions. Descriptive region names help software engineers quickly understand where subconcern code exists in relation to the rest of the program logic. Nonetheless, if a region name provides insufficient contextual information, the user can always click on the name to see that region's context in the unwoven-body panel.

As another example, please consider Figure 5, which shows how IVCon groups into one subconcern all syntactically equal code assigned to the same concern. Figure 6 contains the woven view of the same program. In the woven view, the user has defined a concern named `constant` and has assigned the two constants 512 and 2000 to that concern (IVCon's token-level granularity in concern assignment enables such operations). Normally, programmers using standard software-development tools would define these values in memory declared immutable and would always refer to the constants' values with variables like `MAXBUFFERSIZE` and `TIMEOUTMS`. This technique enables the programmers to make a global change to a constant value by modifying just one central definition. However, this benefit comes at the price of not being able to immediately see the values of constants in the source code. In contrast, IVCon's dual woven and unwoven views provide *both* benefits: users can update constant values centrally (in the `constant` concern of the unwoven-concerns panel) and can view the constant values directly in the source code (in the woven-body panel). Actually, IVCon provides the added benefit that users can see (in the unwoven-concerns panel) a region-name reference to every use of every constant and can click any of those region names to see the context of that use in the unwoven-body panel. Of course, this example is just a special case of the general use of IVCon to provide both a global (i.e., woven) view of the code and a concern-specific (i.e., unwoven) view of the same code.

IVCon's unwoven view allows users to create and edit core-program code (in the unwoven-body panel) and concern code (in the unwoven-concerns panel). To avoid ambiguity in the weaving algorithm, IVCon does not allow users to delete holes in the unwoven-body panel or to edit non-concern code (e.g., concern and region names) in the unwoven-concerns panel. Also, for simplicity, IVCon currently does not allow users to create concerns or to change concern assignments in the unwoven view; these restrictions save IVCon from ever having to simultaneously update noncontiguous code segments in the unwoven-concerns panel.

C. Display of Multi-concern Code

The woven view displays multi-concern code in white text over the multi-concern background, while the concerns-at-current-position panel indicates which concern(s) the code at the current cursor position implements. Similarly, the unwoven view displays multi-concern code in the unwoven-body panel as a hole colored white over the multi-concern background, while the concerns-at-current-position panel continues to indicate which concern(s) the hole at the current cursor position implements.

In addition, the unwoven-concerns panel uses *flags* to convey information about the concerns associated with multi-concern code (as mentioned in Section II-B). Flags serve as a quick reference for visualizing where overlapping concerns begin and end. To illustrate the use of flags, consider the code in Figure 3 that implements the `security` concern at region `before_protected_read`. In the woven view (Figure 2), we assigned this code segment to the `security` concern, and we assigned the two nested statements beginning with `accessLog.append` to the `audit` concern. As a result, the unwoven-concerns panel in Figure 3 displays green flags ($^{\text{audit}}$🟩) and red flags(🟥$^{\text{audit}}$) to indicate the nesting of `audit`-concern code within the `security`-concern code.

Green and red flags within the unwoven-concerns panel indicate the beginning and ending of overlapping concerns. Green and red flags do not always appear together within a subconcern; depending on the overlap between concern regions, there may be a green flag only, a red flag only, both green and red flags, or no flags at all within subconcern code. Also, multiple red and green flags of the same concern, or multiple flags of various concerns, may be present within a subconcern.

The syntax of code in IVCon's unwoven-concerns panel includes flags, so two subconcerns are syntactically equal if and only if the text of those two subconcerns—including flags within the subconcerns—is the same. Thus, the subconcern `accessLog.append(``File read complete.'')` is not syntactically equal to $^{\text{security}}$🟩`accessLog.append(``File read complete.'')`🟥$^{\text{security}}$. This distinction matters because, as described in Section II-B, syntactically equal subconcerns are grouped together in the unwoven-concerns panel. If IVCon did not consider flags when testing for concern-code equality, it would group both `accessLog.append(``File read complete.'')` and $^{\text{security}}$🟩`accessLog.append(``File read complete.'')`🟥$^{\text{security}}$ into a single subconcern, but the presence or absence of security flags in such a subconcern would be confusing because one of the instances of this subconcern has a nested security region, while the other does not.

III. Implementation

We have implemented a prototype of IVCon on a Java platform [11]. The core IVCon application consists of 6235 lines of code, of which 6107 implement the GUI and 128 implement backend data structures. IVCon also uses several third-party libraries (e.g., clipboard and lexical-analysis libraries) for auxiliary functions.

A. Data Structures

IVCon maintains three key data structures.

1) A *regionMap* is a hash table that maps a numerical region identifier to that region's user-visible name, beginning and ending positions for the region, and a list of the concerns to which the region has been assigned.
2) A *concernMap* is a hash table that maps a unique concern name to that concern's display color and a list of the regions assigned to that concern.
3) A *regionTree* is an RTree, a dynamic structure for storing data about potentially overlapping regions in space [12], [13]. When queried about a particular region r, an RTree can efficiently return the set of stored regions that overlap r. RTrees are ideal data structures for IVCon because they enable efficient querying to determine which regions are defined at a given character position in the source file (e.g., at the position of the cursor or within a newly defined region). Once IVCon determines which regions are present at a given position, it can use the `regionMap` to look up and display all the concerns assigned to those regions. IVCon uses Hadjieleftheriou's implementation of RTrees [14].

Together these structures enable reasonable performance in all of IVCon's core operations.

B. Translation Algorithms

Given a `regionMap`, `concernMap`, and `regionTree`, it is generally straightforward to implement the translation from woven to unwoven view, and vice versa.

1) Unweaving: Unweaving is the process of translating a woven-view program into an equivalent unwoven-view program. Unweaving in IVCon begins with lexical analysis to (1) ensure that all tokens in the current program are valid Java tokens and (2) enforce token-level granularity in concern assignment. After lexical analysis, IVCon starts with the woven code in the unwoven-body panel and iterates through all concerns in the `concernMap`, extracting the code in all of that concern's regions from the unwoven-body panel to the unwoven-concerns panel. Extracted concern code gets replaced with holes (□) of the appropriate color in the unwoven-body panel. During the extraction process, IVCon groups concern code into syntactically equal subconcerns in the unwoven-concerns panel and displays isolated concerns in the format described in Section II-B.

Although the unweaving algorithm just described is straightforward, one interesting issue arose during implementation. The issue concerns how to display holes in place of overlapping, but unequal, regions. For example, let us reconsider the woven-body code of Figure 2. In that figure, a programmer has assigned an entire `if` statement to the `security` concern and has nested two `audit`-concern regions within that `security` region. There are two reasonable alternatives for unweaving this `if` statement:

1) We could replace the entire `if` statement with *one* hole of the multi-concern color to indicate that one region of code, which in total implements multiple concerns, has been extracted.
2) We could replace the entire `if` statement with *five* holes that alternate between the security-concern color and the multi-concern color. These holes would indicate that the extracted code first contains security-concern code, then some multi-concern code, then more security-concern code, and so on.

Because it provides more precise concern-assignment information, we implemented the second of these alternatives in IVCon. Figure 3 shows the resulting five-hole concern extraction in the unwoven-body panel.

2) Weaving: Weaving is the process of translating an unwoven-view program into an equivalent woven-view program. To weave, IVCon builds the woven body from the unwoven body by iterating through all the holes in the unwoven body and filling in each hole with the appropriate concern code.

The one interesting problem that can arise during weaving relates to filling multi-concern holes with code. Because a code segment that fills in a multi-concern hole has been assigned to multiple concerns, it appears in multiple places in the unwoven-concerns panel. The problem arises if a user edits this code in one concern and not in all the other concerns where it appears (e.g., a user may have changed the `log.appends` to `log.prepends` in the `security` concern of Figure 4 without modifying the `log.appends` in the `audit` concern). In this case, IVCon cannot immediately determine with what code to fill the hole, so it opens a popup window to (1) notify the user of the concern-code inconsistency and (2) allow the user to select which code segment to fill the hole with (e.g., to use `log.prepend` or `log.append`). An alternative to asking users to select a code segment when an inconsistency is detected would be to avoid the inconsistency altogether by using linked editing [15]. By linking all subconcerns in the unwoven-concerns panel that represent the same hole in the unwoven-body panel, any changes made in one subconcern would be immediately reflected in all the other subconcerns that represent the same hole. We plan to include this feature in a future version of IVCon.

After IVCon fills in all the holes, it has a complete woven view of the program and can perform lexical analysis. Like the lexical analysis at the beginning of the unweaving process, lexical analysis at the conclusion of the weaving process enforces token-level granularity in concern assignment. Although for simplicity of our prototype implementation we have not done so, it would also be useful to integrate a complete Java compiler into IVCon to check, beyond simple lexical analysis, that woven programs are statically valid Java programs. Integrating a complete compiler and virtual machine into IVCon, or integrating IVCon into an existing IDE (e.g., Eclipse), would also save programmers from having to switch to external tools for code compilation and execution.

IV. Performance Evaluation

To evaluate the practicality of our design, we tested our implementation by assigning code to concerns in three of our own IVCon source-code files: `IVCON.java`, `FileUtilities.java`, and `ConcernManipulation.java` (our largest source-code file), respectively containing 49, 313, and 3342 lines of Java code (in the woven view) and 7, 25, and 182 regions assigned to 5, 11, and 36 total concerns. We also created an impractically large file of 100,000 lines, each containing 20 randomly generated single-character tokens, in order to stress test IVCon's performance. We emphasize that `StressTest.java` would be an unreasonably large (2-million-token) source-code file in practice; we included it in our test suite to better understand IVCon's performance limitations.

We measured (on a standard laptop) IVCon's performance weaving and unweaving code, opening, closing, and saving files, and creating, assigning code to, editing, and removing concerns. Interested readers may consult our companion technical report for detailed results of these experiments [16]. To summarize the results, for reasonably-sized files (i.e., `IVCON.java`, `FileUtilities.java`, and `ConcernManipulation.java`), the weaving and unweaving operations took approximately a half a second or less, whereas it took approximately eight minutes to weave and unweave code in `StressTest.java`. Our performance results demonstrate that IVCon's design is sufficiently efficient when operating on reasonably sized files, but IVCon would need additional optimizations to perform tolerably efficiently on extremely large (multimillion-token) files.

V. Conclusion

We conclude by describing future extensions to this work and summarizing.

A. Future Work

The highest-priority future work involves hardening our prototype implementation of IVCon using IVCon itself. We plan to recompile (i.e., bootstrap) IVCon after adding each of the features listed below, so that we can draw the benefits of each feature added when implementing the next.

- Search for text in code. This serves a feature similar to <ctrl-f> in common editors.
- Open multiple files simultaneously. This will aid users in building projects that span over multiple files.
- View flags in the woven view. This will provide users with the same clarity of concern assignment in multi-concern code in the woven view that currently exists in the unwoven view.
- Show in a tool-tip the concerns implemented by the code that the mouse pointer points to. This will enable users to see which concerns a code segment S implements without actually moving the cursor to S.
- Integrate linked editing in the unwoven-concern panel in the unwoven view (as mentioned in Section III-B2). This will ensure that during weaving there are no inconsistencies in the code corresponding to multi-concern holes.

We have omitted all of these features from the current prototype implementation because their inclusion does not appear to offer much in the way of new research insights; that is, none of these features would contribute to IVCon's novelty. Rather, our motivation for implementing these features is to provide ourselves with additional experience using IVCon.

B. Summary

We have presented IVCon, a tool for conveniently creating, examining, and modifying code in, and translating between, woven and unwoven views of code. IVCon differs from existing aspect-visualization tools by providing all the features summarized in Figure 1. IVCon has a generally straightforward interface and a prototype implementation online [11]. In all of our empirical tests, which involved assigning concerns to, and modifying, real IVCon source code, IVCon performed efficiently on all practical source-code files.

References

[1] P. Tarr, H. Ossher, S. M. Sutton, Jr., and W. Harrison, "N degrees of separation: Multi-dimensional separation of concerns," in *Proceedings of the International Conference on Software Engineering*, 1999, pp. 107–119. [Online]. Available: citeseer.ist.psu.edu/tarr99degrees.html

[2] T. Panas, J. Karlsson, and M. Högberg, "Aspect-jEdit for inline aspect support," in *Proceedings of the German Workshop on Aspect Oriented Software Development*, 2003.

[3] "The Visualiser," 2008, http://www.eclipse.org/ajdt/visualiser/. [Online]. Available: http://www.eclipse.org/ajdt/visualiser/

[4] W. G. Griswold, J. J. Yuan, and Y. Kato, "Exploiting the map metaphor in a tool for software evolution," in *Proceedings of the International Conference on Software Engineering*, 2001, pp. 265–274.

[5] M. Marin, L. Moonen, and A. van Deursen, "SoQueT: Query-based documentation of crosscutting concerns," in *Proceedings of the International Conference on Software Engineering*, 2007, pp. 758–761.

[6] M. P. Robillard and G. C. Murphy, "FEAT: a tool for locating, describing, and analyzing concerns in source code," in *Proceedings of the International Conference on Software Engineering*, 2003, pp. 822–823.

[7] C. Kästner, "CIDE: Decomposing legacy applications into features," in *Proceedings of the International Software Product Line Conference, second volume (Demonstration)*, 2007, pp. 149–150.

[8] C. Kästner, S. Apel, and M. Kuhlemann, "Granularity in software product lines," in *Proceedings of the International Conference on Software Engineering*, May 2008, pp. 311–320.

[9] H. Ossher and P. Tarr, "Hyper/J: Multi-dimensional separation of concerns for Java," in *Proceedings of the International Conference on Software Engineering*, 2000, pp. 734–737.

[10] M. Yuen, M. E. Fiuczynski, R. Grimm, Y. Coady, and D. Walker., "Making extensibility of system software practical with the C4 toolkit," in *Proceedings of the Workshop on Software Engineering Properties of Languages and Aspect Technologies*, March 2006. [Online]. Available: http://www.cs.princeton.edu/~dpw/papers/splat06.pdf

[11] N. Saigal and J. Ligatti, "IVCon – Inline Visualization of Concerns," 2008, http://www.cse.usf.edu/~ligatti/projects/ivcon/.

[12] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47–57.

[13] L. Arge, M. D. Berg, H. Haverkort, and K. Yi, "The priority R-tree: A practically efficient and worst-case optimal R-tree," *ACM Trans. Algorithms*, vol. 4, no. 1, pp. 1–30, 2008.

[14] M. Hadjieleftheriou, "Spatial index library," http://www.research.att.com/~marioh/spatialindex/index.html.

[15] M. Toomim, A. Begel, and S. Graham, "Managing duplicated code with linked editing," in *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*, Sept. 2004, pp. 173–180.

[16] N. Saigal and J. Ligatti, "Defining and visualizing many-to-many relationships between concerns and code," University of South Florida, Tech. Rep. CSE-090608-SE, Sep. 2008. [Online]. Available: http://www.cse.usf.edu/~ligatti/papers/ivcon-tr.pdf