International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:9, 2008

# An Approach to Concerns and Aspects Mining for Web Applications

Carlo Bellettini, Alessandro Marchetto, and Andrea Trentini

***Abstract*—**Web applications have become very complex and crucial, especially when combined with areas such as CRM (Customer Relationship Management) and BPR (Business Process Reengineering), the scientific community has focused attention to Web applications design, development, analysis, and testing, by studying and proposing methodologies and tools. This paper proposes an approach to automatic multi-dimensional concern mining for Web Applications, based on concepts analysis, impact analysis, and token-based concern identification. This approach lets the user to analyse and traverse Web software relevant to a particular concern (concept, goal, purpose, etc.) via multi-dimensional separation of concerns, to document, understand and test Web applications. This technique was developed in the context of WAAT (Web Applications Analysis and Testing) project. A semi-automatic tool to support this technique is currently under development.

***Keywords*—** Aspect Mining, Concepts Analysis, Concerns Mining, Multi-Dimensional Separation of Concerns, Impact Analysis.

## I. Introduction

WEB applications quality, reliability and functionality are important factors because software glitches could block entire businesses and determine strong embarrassments. These factors have increased the need for methodologies, tools and models to improve Web applications (design, analysis, testing, and so on).

This paper focuses on legacy Web applications where business logic is embedded into Web pages. Analyzed applications are composed by Web documents (static, active or dynamic) and Web objects [6]. This paper describes an approach to help application developers to document, understand and test Web software. Our goal is to describe a Web application Object-Oriented model, and then define a set of application/design slices ("points of view") to analyze and test the application itself, e.g., to generate a set of test cases specific for these points of view. Several Object-Oriented Web modeling methodologies are presented in literature (see Section II). Web OO diagrams (such as Conallen UML [12]) used to describe applications may be very complex, large, and rich of information. Models (above all generated ones) may be difficult to read and comprehend, so that they may not be much usable as core information to document, analyze and test applications. Our approach may be useful to slice or traverse models for software analysis. For example, it may be very interesting to test or reuse single components or tasks or properties, but it may be very complex to spot the relevant details within the whole design documentation. Software concerns are pieces of software that are responsible for a particular task, concept, goal, etc; while "separation of concerns" refers to the ability to identify, encapsulate and manipulate those software parts relevant to a particular concern.

This paper describes a semi-automatic approach to help the user to document, understand and test Web software by slicing applications diagrams. Application model slicing is based on concerns identification and grouping. Our approach describes a set of guidelines to analyze application evolution under different "points of view" (i.e., slices). In particular we would like to define a concern-mining process to help the user to generate application test cases and/or to verify their coverage measure. Our approach is useful to identify multi-dimensional concerns (MDSOC, [20],[36]) in design applications, it uses the MDSOC "dimensions of Hyperspace" concept to describe application slices in Web software. "Hyperspace" is the concept underlying MDSOC, it provides a powerful composition mechanism that facilitates non-invasive software integration and adaptation. In Hyperspaces, concerns are space dimensions. Our concerns mining approach is based on: concepts analysis[1] [17] (as unit-base to identify concerns); impact analysis [2] (to limit software analysis); and token-based concerns identification (to search identified information relationship). This technique is part of the WAAT (Web Application Analysis and Testing) project [6],[5].

This paper is organized as follows. Section II describes related works. Section III describes applications modeling. Section VI introduces our reverse engineering techniques to model recovery. Section V presents some background. Section VI describes our concerns mining approach. Section VII presents a sample. Section VIII Section IX analyses a case study. Finally, Section X presents conclusions.

Manuscript received March 12, 2005.

Carlo Bellettini, Alessandro Marchetto, and Andrea Trentini are with Information and Comunication Department, University of Milan. Via Comelico 39, 20135 Milan, Italy.
{Carlo.Bellettini, Alessandro.Marchetto, Andrea.Trentini}@unimi.it

[1] Concept analysis is "traditionally" used to show all possible software modularizations in a concise *lattice* structure

## II. Related Works

Several Web applications **modeling** approach are presented in literature [6]. RMM [21] is a method based on Entity-Relationship diagrams, and is specialized in applications based on databases. WebML [11] enables the description of a Web site under distinct orthogonal dimensions (such as structural, composition, and so on). [28] introduces a Web application simulation model framework that was designed to be compatible with some existing modeling languages. Often, these web methodologies are extensions of traditional methodologies, such as OOHDM [33] for Object Oriented ones. It uses OO models to define: conceptual, navigational and user interface structure of applications. Moreover, some of these are UML based. WARE [14] and Rational Rose Web Modeler [30] are tools for reverse engineering supporting Conallen's extensions [12]. Both tools perform essentially static analyses to generate model. Our WebUml [6] is tool to reverse engineering Web application through static and dynamic analysis. These OO modeling approach derived are related to our concerns mining technique.

Some currently available Web **testing** tools (e.g. [25]) are usually classifiable as syntax validators, HTML/XML validators, link checkers, load/stress testing tools, and regression testing tools, i.e., they are focused on low-level (implementation bound) or non-functional aspects. Some of these tools are often based on capturing user gestures and replay them through testing scripts. These tools cannot provide structural or behavioral test artifacts. Moreover, they represent a good compromise when a formal model is not available and the only implicit model is the user itself.
Other existing tools, such as xUnit (e.g, [19]), propose a different approach, based on unit/functional testing. Other approaches based on functional, structural and behavioral OO model testing, are: [14], [5], [24], [32]. [14] proposes a strategy to build functional unit-integration testing based on WARE described model. [24] proposes an OO Test Model that captures artifacts representing objects, behaviors, and structure aspects. From this model, structural and behavioural test cases may be derived to support the test process. [32] describes tools: ReWeb, performing several traditional source code analysis, to reverse engineering Web applications into UML model; TestWeb, that uses ReWeb models to test applications through Web site validation paths. [5] describes TestUml tool for XML-based test cases generation derived from WebUml extracted model. [22] defines statistical testing based on usage model described from log files and then analysed with Unified Markov Models.

More details about **Aspect Oriented programming** are in [23], while [3] presents the AspectJ famous software. [20],[36] describe the **MDSOC** and HyperJ tool, while [27] studies the relations between quality factors and MDSOC, while [35] the relations between MDSOC and testing. [32] describes our approach to apply Multi-Dimensional Separation of Concerns (MDSOC) theory at Web applications. [31] describes SOC used to reduce the complexity of Web applications. [18] presents an approach to separate Web navigation concerns and application structure. [9] evaluates AOSD code quality influence and presents an approach for reverse engineering aspects, based on concern verification and aspect construction. [10] evaluates the suitability of clone detection as a technique for the identification of crosscutting concerns via manual concern identification. [13] introduces aspect mining and identification in OO. [8],[37] show an approach to aspect mining based on dynamic analysis technique via program traces investigation, to search recurring execution relations. [26] applies three different separation of concerns (SOC) mechanisms (HyperJ, AspectJ, and a lightweight lexically based approach) to separate features in the two software packages. This paper studies effects that various mechanisms have on code-base structure and on restructuring process required while performing separations.

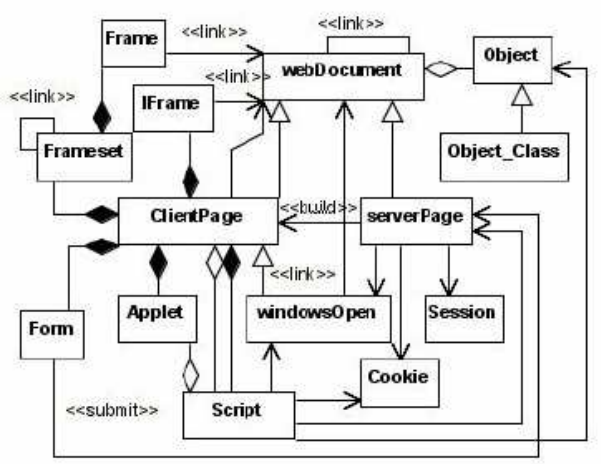


**Figure 1: UML Class diagrams Meta-Model**

## III. Web Applications Modeling

In the WAAT project Web applications are modelled via UML diagrams [6]. The UML model used is based on class and state diagrams. We have defined a UML meta-model (Figure 1: UML Class diagrams Meta-Model), a Web application model is an instance of this meta-model. Class diagrams are used to describe application structure and components (i.e., forms, frames, Java applets, HTML input fields, session elements, cookies, scripts, and embedded objects). State diagrams are used to represent behaviour and navigational structures composed by client-server pages, navigation links, frames sets, form inputs, scripting code flow control, and so on.

The OO application model let us define a mapping between traditional Web application concepts (such as static-dynamic pages, forms, Web objects, and so on) and the MDSOC concepts. This map let us apply separation of concerns methodologies in the Web context, for example to analyse or

test specific assets of existing software. Our approach may be used to "slice" application models by "points of view". This approach is useful with our Web modelling technique, but it may be useful with every other OO modelling techniques applied to Web software (such as presented in previous Section).

## IV. Model Recovery

Our approach ([6],[7]) to model recovery is composed by: **application behavior** analysis, application **model building,** and **model validation**.

**Application behavior** analysis is performed through static and dynamic analysis. Static and dynamic analyses treat static and dynamic application components using source code and on-line interactions with the Web server. For example, for static pages, we use traditional source code analysis based on a language parser. While, for a single server page generating multiple client pages, we apply dynamic analysis to try to determine a meaningful number of client pages (through mutation analysis and application executions). Then, the dynamically generated client–side pages are analyzed (with traditional source code analysis) to build diagrams. More generally, for every dynamic Web document, we use mutation analysis to define mutants (for example changing the control–flow structure of original source code page) to be fed into session navigation simulations, in which every mutant replaces the original source code and the simulation performs generated interactions. This simulation is used to send input values and page requests to the Web server, and saving responses that are analyzed later.

Mutation analysis is based on mutant operators applied to source code, and in particular to control–flow source code fragments (e.g., "if-then else", "while", etc.), such as logic or Boolean operators, conditions or check operators, and so on. For example, the "=" operator can be mutated into "<>", the ">" operator can be mutated into "≤" or "<", the "AND" operator can be mutated into "OR", etc. The aim of mutation is to automatically follow relevant execution paths in the Web application, to cover as many navigation paths as possible.

This approach does not need knowledge about the language, only a simple map of mutant operators, deployable with easy to program parsers and with low computational complexity.

**Model building**; with the information extracted by the previous phase we build an application OO model (such as described in the previous section) using UML class and state diagrams.

**Model Validation**; The "mutation" generated model may contain more information than what is needed. In particular, it may contain "Not-Valid" information, such as not valid dynamically generated client-side pages. A client-page is "Valid" if it is reachable in the original application (without mutants) via an execution path. Since mutation may define a model with a super-set of behaviors we need a pruning technique. Our proposed technique is essentially based on Web server log files analysis validation and "Visual Navigation validation" with the user help.

In particular we analyze the Web server log files (e.g., the Apache Web server log files in Figure 2: Fragment of Web Server Log File) and we replay every Web request (for dynamic pages) to analyse the server response. We match these responses with pages in model (introduced using mutation analysis). The set of matched pages are the "Verified" pages. Every "Verified" page exists in original application. For every "Not-Verified" page we ask the user help to validate it. Via model analysis we may define a set of paths containing the "Not-Verified" pages (every ones for a path). User may mark "Valid" a page in a path, if the page is reachable through that path in the original application (without mutants).

```
127.0.0.1 – [26/May/2004:18:04:02 +0200]
   "GET /website/index.html HTTP/1.1" 200 1560
127.0.0.1 – [26/May/2004:18:05:52 +0200]
   "GET /website/dynamicP.asp?code=056978&name=Alex
   HTTP/1.1" 200 1802
127.0.0.1 – [26/May/2004:18:7:26 +0200]
   "GET /website/clientP2.html HTTP/1.1" 200 1727
127.0.0.1 – [26/May/2004:18:7:59 +0200]
   "GET /website/index.asp HTTP/1.1" 200 700
127.0.0.1 – [26/May/2004:19:02:10 +0200]
   "GET /website/pageResource.html HTTP/1.1" 200 2563
```

**Figure 2: Fragment of Web Server Log File**

The proposed approach is useful to describe existing Web applications via a UML model built in is semi-automatic way. Model construction is automated via mutation analysis, while model validation is quite user dependent. Traditional ways to analyze existing Web software focus on application source code analysis of control-flow expressions to identify representative inputs values. Inputs values are used to define feasible application behaviors. In this conventional approach user must know the application language and must know the control-flow concepts and condition control analysis. The use of mutation analysis decreases user interactions needed to build application models, because mutation changes the analysis perspective, from source code analysis to application analysis. The model may contain spurious information and must be pruned and validate (via Model Validation approach).

## V. Background

Our proposed approach to concerns mining is based on MDSOC and Concept Analysis. In this section we briefly introduce these theories.

**MDSOC** (Multi-Dimensional Separation of Concerns) is an approach to implement separation of concerns (SOC) by IBM. IBM implemented a tool named Hyper/J to support MDSOC in Java software. Separation of concerns refers to the ability to identify, encapsulate, and manipulate software fragments relevant to a particular concern (concept, goal, purpose, etc.). Concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts. MDSOC lets the user model applications via multi-dimensional structure (named Hyperspace), instead of other OO-techniques that model applications by only one dimension ("tyranny of the dominant decomposition").

Moreover MDSOC encapsulates many kinds of concerns at the same time, and models overlapped concerns and concerns interaction. MDSOC is very useful for on-demand software re-modularization.

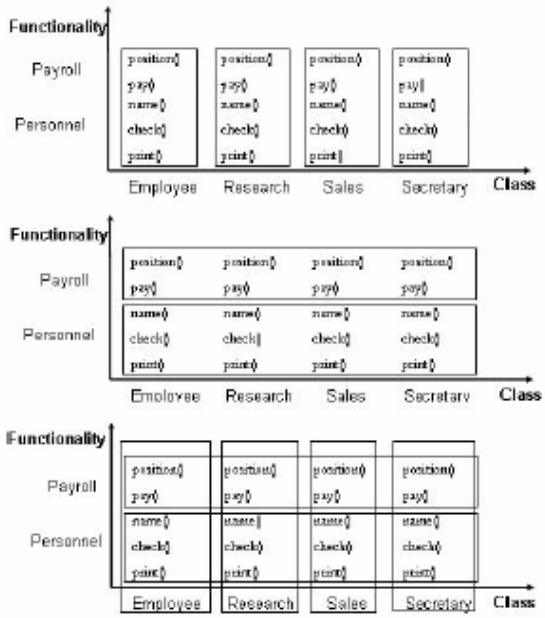


**Figure 3: MDSOC hyperspaces sample**

Figure 3: MDSOC hyperspaces sample, (see [29]) shows Hyperspace samples for an example personnel software system, these Hyperspaces are composed by two dimensions, axes are software *Class* (e.g., *Employee*, *Research*, *Sales*, *Secretary*) and interesting *Functionality* (*Payroll*, *Personnel*), while points in space are software units, such as class methods (or statements). In case of "dominant tyranny" (OO or Aspect modelling [36]) only one concern type is encapsulated (e.g., first/second plane in Figure 3, where only class or functionality are encapsulated). Instead, MDSOC supports clean separation of multiple, overlapped and interlaced concerns simultaneously, and on-demand re-modularization (e.g., the third plane in Figure 3: MDSOC hyperspaces sample, shows the on-demand re-modularization for system class and functionality).

**Formal Concept Analysis** (FCA, [15]) is a theory of data analysis which identifies conceptual structures among data sets. Concept Analysis is applied to many fields, such as medicine and psychology, musicology, linguistic databases, library and information science, software re-engineering, civil engineering, ecology, and others.

Concept Analysis important capability is the graphical visualization of these structures among data via the *concept-lattice*. Lattices may be interpreted as classification systems. For example in software engineering, FCA may be useful to show all possible software modularizations in only one concept-lattice or to re-modularize software (e.g., introducing "aspects" in OO existing software).

Concept analysis provides a way to identify grouping of *objects* that have common *attributes.* Given a context=(O, A, R), where: O=objects, A=attributes, R=binary relation between O and A, we may use the concept-analysis grouping algorithm to define concepts. Concepts are "the maximal collection of objects sharing common attributes".

Default Name

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| Default Name | Four-legged | Hair-covered | intelligent | marine | thumbed |
| cats | X | X | 0 | 0 | 0 |
| chimpanzees | 0 | X | X | 0 | X |
| dogs | X | X | 0 | 0 | 0 |
| dolphins | 0 | 0 | X | X | 0 |
| humans | 0 | 0 | X | 0 | X |
| whales | 0 | 0 | X | X | 0 |

**Figure 4: Context-Matrix sample**

For example (see [34]), Figure 4: Context-Matrix sample, shows a generic context matrix with couples of "object-attribute", where objects are living beings, while attributes are five of their possible characteristics. For example:

({cats, dogs}, {four-legged, hair covered}) is a concept.
({cats, chimpanzees}, {hair covered}) is not a concept.

Then, we may define a relationship via hierarchical organization of the defined concepts by describing the relative concept-lattice (in Figure 5: Concept-Lattice sample). Last, by applying the FCA algorithm the concepts table (see Table I) is built.

TABLE I
CONCEPTS SAMPLE

| | |
|---|---|
| top | ({*..all..*}, ø) |
| C5 | ({chimpanzees, dolphins, humans, whales}, {intelligent}) |
| C4 | ({cats, chimpanzees, dogs}, {hair-covered}) |
| C3 | ({chimpanzees, humans}, {intelligent, thumbed}) |
| C2 | ({dolphins, whales}, {intelligent, marine}) |
| C1 | ({chimpanzees}, {hair-covered, intelligent, thumbed}) |
| C0 | ({cats, dogs}, {hair-covered, four-legged}) |
| bot | (ø, {*..all..* }) |

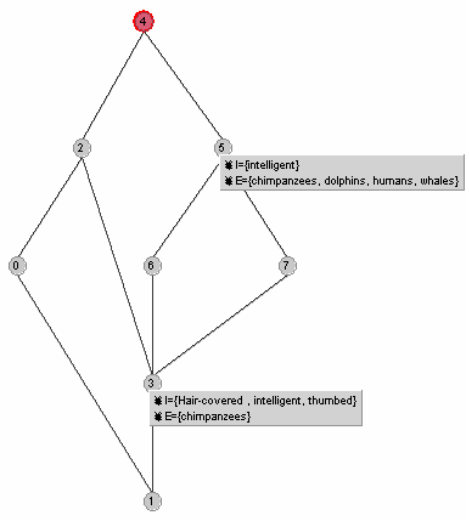


**Figure 5: Concept-Lattice sample**

## VI. CONCERNS DEFINITION ALGORITHM

MDSOC technique is used to build application slices, where every concern (or concerns composition) may be used to define a software code/design slice. MDSOC is realized through Hyperspaces: concerns space organized in multi-dimensional structure. In this structure every dimension is a set of disjoint concerns (i.e., they have no software units in common). We define a semi-automatic concerns mining approach, so concerns identification must be limited to information extracted from applications models or source code. For example, software functionality identification is a semi-automatic task, because the user helps to identify software components. We may lower user interactions by only applying MDSOC to concerns that are automatically extracted. When functionality information cannot be automatically identified, than we use: variables, functions, class, Web documents/objects, links, input-variables, and so on.

Our approach is composed by: **Application Modeling** (**AM**, model definition), **Concerns Elaboration** (**CE**, Hyperspaces definition through model and source code analysis, and Hyperspaces use to reduce models and code taken into account), **Testing** (**T**, the extracted and reduced information may be used to define/refine test cases).

In next Section we present a simple case study useful to describe the proposed approach.

**Application Modeling** (**AM**) consists in application model definition. We use reverse engineering techniques to define UML diagrams for existing applications. Moreover, diagrams may also be manually refined by the user.

**Concerns Elaboration** (**CE**) to identify, define, and extract concerns based on application model or source code analysis, subdivide in:

- *Artifacts extraction*: from application model we extract some interesting artifacts such as class, association, variables, methods, links, Web pages (e.g., static, dynamic, dynamically generated), objects (e.g., database, files, reused code), and so on. We use this knowledge to identify concerns (it may be a limitation, i.e., concerns about functionality cannot be completely defined without user know-how).
- *Objects-attributes selection*: from the selected artifacts we define "object-attribute"[2][17] couples to use in concept analysis. We may limit the number of couples by asking user help. Generally speaking, example of couples may be: variables-classes, instance variables-classes, variables-functions, instance variables-functions, and so on. To select couples user may know concerns/aspects theory, and how define a concern/aspect using classes, variables, functions, and so on. To this task we have defined a set of rules, such as, to define aspect in OO software we may analyze the instance variables used by software functions, and if a function uses variables defined in more than one class, this is a candidate to define a crosscutting concerns (aspects).
- *Impact matrix definition*: from the application model we define a matrix I = [ class x class ]. $\forall\ i_{k,m} \in I = 1$ if $\exists$ class relationship (i.e. association in class diagram between $class_k$ and $class_m$), 0 otherwise. The matrix is then used to decrease analysis computational cost.
- *Context matrix definition*: for every couple defined we build an objects-attributes matrix C = [object x attribute]. $\forall\ c_{k,m} \in C = 1$ if there is a def-use relationship between $object_k$ and $attribute_m$, 0 otherwise.
- *Concept definition/visualization*: we define concepts through the C contexts matrix. We analyze this matrix grouping the maximum number of objects that have common attributes (by concept definition in concept-analysis). To visualize the defined concepts we may use the concept-lattice [17] structure, and in particular we may use existing software such as ToscanaJ [38] or Galicia [16].
- *Concerns identification:* we may identify concerns by the concepts defined in the concept lattice structure. Every node in the lattice is a concept (by concept-analysis definition). Every concept is a concern. Concept is groups of "objects" (in concept analysis sense) that sharing "attributes" (in concept

[2] where "objects-attributes" is defined in concept-analysis theory

analysis sense). For example, for the "object-attribute" couple "variables-classes", concepts are groups of variables that sharing definition or use classes. In this example, code fragments using a common set of variables may represent candidates to identify software behaviors.

- *Concerns composition*: to compose the defined concerns we must analyze the concerns dependencies to define autonomous behaviors or behaviors dependencies. To this aim we propose two ways. With the first approach we may traverse the lattice structure based on the concepts dependencies (associations between nodes in lattice structure). For example, from bottom to top nodes to identify concept-objects dependencies, while from top to bottom nodes to identify concept-attributes dependencies. With the other approach we identify concerns by iteratively grouping previously defined concepts. We define a new "attributes-concepts" matrix[3] A = [ attribute x concept ]. $\forall\ a_{k,m} \in A = 1$ if *attribute* is contained in *concept*. By recursively applying the "attributes-concepts" matrix, at each step we build supersets of concepts (grouping concepts that share attributes) that are used as concepts as well in the next step. In this way is possible to group concept/concerns that share information. Every information-sharing between concerns represents a concerns dependencies candidate.

**Testing** (**T**): to slice applications. The defined concerns may be useful to slice application models or source code. In particular, we would like to use extracted information (application slices) to define test cases and to compute application coverage level for a set of already available test-cases. For example, we may define test cases from a UML model (e.g., from a statechart, see Section III) via traditional OO techniques and then use the reduced diagram to verify test-cases coverage (e.g., uniformly coverage or specialized one). Otherwise we may define test-cases directly from the reduced diagrams, because they represent sets of application features (software fragment with potentially independent behavior).

## VII. Sample

"*MiniLogin*" (Figure 6: MiniLogin application Home-Page) is a simple Web application composed by some PHP/HTML files, and its main functionality is to control reserved login-password Web area.

**Figure 6: MiniLogin application Home-Page**

[3] where *attribute* is from the C matrix, and *concept* was defined in the previous "Concept definition" step

**Application Modeling** (**AM**): we reverse engineer (through the approach proposed in Section IV) the application UML model, composed by class and state diagrams. Figure 7: MiniLogin UML Class diagram, shows the generated application class diagram (meta-model instance).

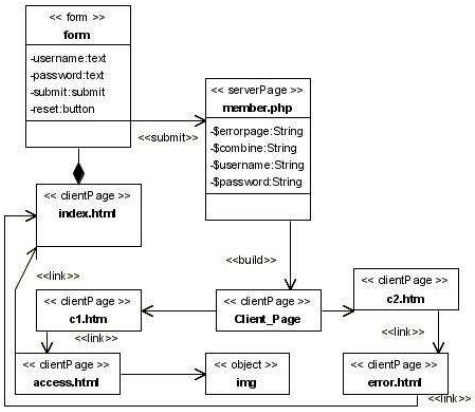


**Figure 7: MiniLogin UML Class diagram**

**Concerns Elaboration** (**CE**): defines MiniLogin concerns.

*Artifacts extraction:* we extract MiniLogin artifacts, lists of: classes, variables, functions, links, and so on.

- *Objects-attributes selection:* we manually select couples of objects attributes to use in concept analysis. E.g., variables-class (named "case-A"), variables-functions, and so on.
- *Impact matrix definition* (due to lack of space we exemplify only a couple of entries): "form" is related to "member.php", while "form" is not related to "C2.html".
- *Context matrix definition* (due to lack of space we exemplify only a couple of entries): for "case-A" "$errorpage" is related to "member.php", "username" is related to "form", and "username" is related to "member.php", while "username" is not related to "C2.htm"

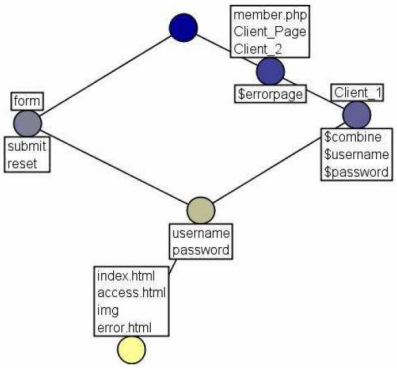


**Figure 8: Concept Lattice for MiniLogin "case-A"**

- *Concept definition/visualization:* we use the context matrix to define concepts as defined in formal concept-analysis [9]. We may use existing tools (such as ToscanaJ [38], to define and visualize concepts through concept-lattice). Figure

8: Concept Lattice for MiniLogin "case-A", shows the concept lattice for "case-A" concepts.

TABLE II
CASE-A, CONCEPTS

| Concept | Object | Attribute |
|---|---|---|
| Top | *...all...* | - |
| C3 | {username, password, $errorpage, $combine, $username, $password} | {member.php, Client_Page, c1.htm} |
| C2 | {username, password, $combine, $username, $password } | {member.php, Client_Page, c2.htm, c1.htm } |
| C1 | {username, password, submit, reset} | {form } |
| C0 | {username, password} | {form, member.php, Client_Page, c1.htm, c2.htm } |
| Bottom | - | *...all...* |

- *Concerns identification:* we identify concerns based on the defined concepts. Every lattice node is a concern. Table II shows "case-A" concepts.

TABLE III
CASE-A, "ATTRIBUTES-CONCEPTS" MATRIX

| Attribute | C0 | C1 | C2 | C3 |
|---|---|---|---|---|
| index.html | | | | |
| form | 1 | 1 | | |
| member.php | 1 | | 1 | 1 |
| Client_Page | 1 | | 1 | 1 |
| c1.htm | 1 | | 1 | 1 |
| c2.htm | 1 | | 1 | |
| img | | | | |
| access.html | | | | |
| error.html | | | | |

- *Concerns composition:* we compose concerns via concepts grouping. We build the attributes-concepts matrix, with attributes used (rows) and concepts (columns). A cell is = 1 if the attribute is related to the concept (see Table III). Then we group concepts by looking for attributes sharing (in our "case-A", variables). E.g., for "case-A" we group concepts into Z0-to-Z4 groups. Where Z0={C0,C1}; Z1={C0,C2}; and Z2/3/4={C0,C2,C3}.

Now we repeat the attributes-concepts matrix definition, using the same attributes list, but with the newly-grouped concepts (Z0-to-Z4) and then we group these concepts attributes-based defining other new concepts (called ZZ0-to-ZZ4). Then we stop because these concepts are completely overlapped. Finally, we may define the set of composed concerns, where every Cx, Zx and ZZx is a good candidate (usable for our testing task). To reduce the number of candidates we delete overlapped concerns (see Table IV). Every defined concern represents a clearly defined software behavior candidate. We use these concerns to describe the Hyperspace slicing our application, and define the reduced diagrams.

TABLE IV
CASE-A, CONCERNS ENCAPSULATION, ALL ITERATIONS

| Concept | Object | Attribute |
|---|---|---|
| C3 | {username, password, $errorpage, $combine, $username, $password} | {member.php, Client_Page, c1.htm} |
| C2 | {username, password, $combine, $username, $password } | {member.php, Client_Page, c2.htm, c1.htm } |
| C1 | {username, password, submit, reset} | {form } |
| C0 | {username, password} | {form, member.php, Client_Page, c1.htm, c2.htm } |
| Z2 | {username, password, $errorpage, $combine, $username, $password} | {form, member.php, Client_Page, c1.htm, c2.htm} |
| Z0 | {username, password, submit, reset} | {form, member.php, Client_Page, c1.htm, c2.htm } |
| ZZ0 | {username, password, $errorpage, $combine, $username, $password, submit, reset} | {form, member.php, Client_Page, c1.htm, c2.htm } |

**Testing** (**T**): from the reduced diagrams we may automatically define test cases or we may use these diagrams to verify coverage measures of already available test cases (such as in the user metrics driven test cases definition process [5]).

## VIII. CASE STUDY

Figure 9: Home-Page *MiniWP* application, shows the case study application selected for experiment. This application is mini Web portal that functions as news reader, images viewer, and Web reserved area control. MiniWP is composed by twenty PHP/HTML files and few TXT "database" files. Figure 10: *MiniWP* Class Diagram, shows the MiniWP class diagram recoved by our reverse engineering approach.

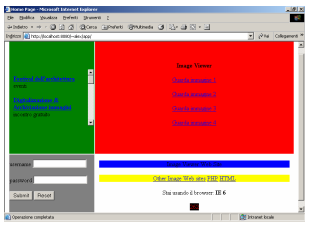

**Figure 9: Home-Page *MiniWP* application**

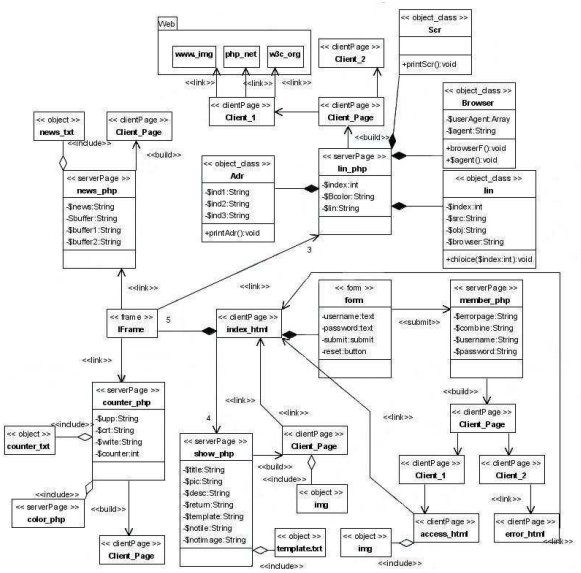

**Figure 10: *MiniWP* Class Diagram**

To apply our concerns mining technique, we select variable-class as *meta*-couple of "object-attribute" (to define class level crosscutting concerns). Then we complete the related *Context-Matrix,* where every cell is 1 if exist def-use relationship between variable and class (if variable is used or defined in class). Then we build the concept lattice related to our selected *meta*-couple. Figure 11: *MiniWP* "variable-class" concept lattice, shows the concept-lattice for MiniWP variable-class *meta*-couple. Based on this lattice we may define the 27 concepts for variable-class *meta*-couple. These are the defined concerns for our case. Then we may define the "attributes-concepts" matrix, we may perform attribute grouping operation, and we may iterate these two last steps to compose the concerns (Figure 12: MiniWP "variable-class" concerns, shows the iteration last step).
Figure 13: MiniWP Class Diagram "marked", shows the class diagram marked with concerns composed that let us slice the model.

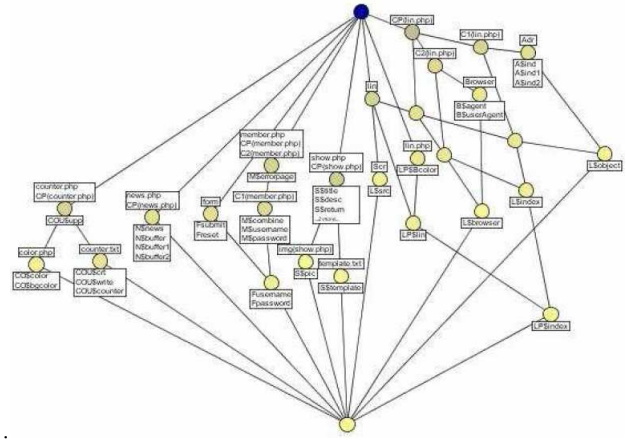

**Figure 11: *MiniWP* "variables-class" concept lattice**

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| index.html | | | | | |
| show_php | | | | 1 | |
| Client_Page(show_php) | | | | 1 | |
| img(show_php) | | | | 1 | |
| template.txt | | | | 1 | |
| Web (Package) | | | | | |
| Iframe | | | | | |
| counter_php | 1 | | | | |
| counter_txt | 1 | | | | |
| color_php | 1 | | | | |
| Client_Page(counter_php) | 1 | | | | |
| news_php | | | 1 | | |
| Client_Page(news_php) | | | 1 | | |
| news_txt | | | | | |
| form | | 1 | | | |
| member_php | | 1 | | | |
| Client_Page(member_php) | | 1 | | | |
| Client_1(member_php) | | 1 | | | |
| Client_2(member_php) | | 1 | | | |
| access_html | | | | | |
| img(access_html) | | | | | |
| error_html | | | | | |
| lin.php | | | | | 1 |
| Lin | | | | | 1 |
| Adr | | | | | 1 |
| Scr | | | | | 1 |
| Browser | | | | | 1 |
| Client_Page(lin_php) | | | | | 1 |
| Client_1(lin_php) | | | | | 1 |
| Client_2(lin_php) | | | | | 1 |

**Figure 12: *MiniWP* "variables-class" iterated concerns**

## IX. Conclusions

We proposed a semi-automatic multi dimensional concerns mining approach based on: concept analysis combined with a grouping technique. This approach may help the user in slicing applications via model analysis, and it may be used to semi-automatically define application test cases, or test coverage measures or also to understand software evolution. We are currently investigating efficient pruning techniques to reduce the number of concerns generated by our approach. We are also working on a tool to integrate our approach in the WAAT project.

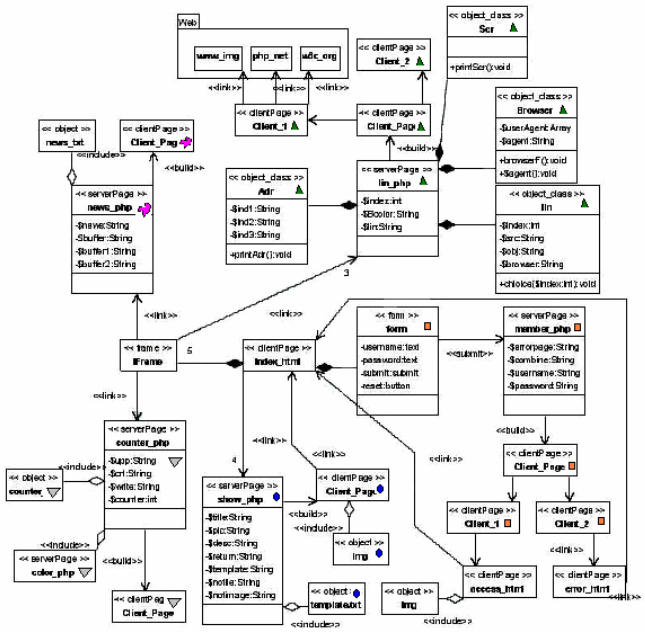

**Figure 13: MiniWP Class Diagram "marked"**

## REFERENCES


[1] Apache Web Server – log file, http://httpd.apache.org/docs/logs.html

[2] T. Apiwattanapong, A. Orso and M.J. Harrold, "Efficient and Precise Dynamic Impact Analysis Using Execute-After Sequences*" 27th IEEE and ACM SIGSOFT International Conference on Software Engineering (ICSE 2005).* USA. 2005

[3] Aspectj. http://eclipse.org/aspectj

[4] C. Bellettini, A. Marchetto, and A. Trentini, "Applying MDSOC to Web Applications" Accepted for publication – *9th World Multi-Conference on Systemics, Cybernetics and Informatics.* Orlando, Florida, USA. July 2005

[5] C. Bellettini C., A. Marchetto, and A. Trentini, "TestUml: User-Metrics Driven Web Applications Testing" 20th ACM Symposium on Applied Computing. USA 2005

[6] C. Bellettini C., A. Marchetto, and A. Trentini, "WebUml: Reverse Engineering of Web Applications". *19th ACM Symposium on Applied Computing (SAC 2004)*, Nicosia, Cyprus. March 2004

[7] C. Bellettini C., A.Marchetto, and A. Trentini, "Validation of Reverse Engineeered Web Application Model." *2th World Enformatika Conference* (WEC 2005). Istanbul, Turkey. February 2005

[8] S. Breu and J. Krinke. "Aspect Mining Using Event Traces". *19th. Conference on Automated Software Engineering 2004 (ASE 04)*, Linz, Austria. September 2004

[9] M. Bruntink, A. van Deursen, and T. Tourwè "An Initial Experiment in Reverse Engineering Aspects from Existing Applications". *11th IEEE Working Conference on Reverse Engineering (WCRE 04)*, Netherlands. November 2004

[10] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwè, "An Evaluation of Clone Detection Techniques for Identifying Cross-Cutting Concerns". *IEEE International Conference on Software Maintenance (ICSM 04)*, 2004

[11] S. Ceri, P. Fraternali, and A. Bongio. "Web Modeling Language (WebML): a modeling language for designing Web sites." *Ninth International World Wide Web Conference (WWW9)*, Amsterdam, Netherlands. May, 2000

[12] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2000

[13] A. Deursen, M. Marin, and L. Moonen, "Aspect Mining and Refactoring". *First International Workshop on REFactoring: Achievements, Challenges, Effects (REFACE03)*, Canada. November 2003

[14] G. A. Di Lucca, A. Fasolino, F. Faralli, and U. De Carlini, "Testing web applications". *International Conference on Software aintenance (ICSM'02)*, Montreal, Canada. October 2002

[15] Formal Concept Analysis, http://www.upriss.org.uk/fca/fca.html

[16] Galicia, http://www.iro.umontreal.ca/~galicia

[17] B. Ganter and R.Wille, "Formal Concept Analysis*". Springer-Verlag*, Berlin, Heidelberg, New York, 1996

[18] M. Han and C. Hofmeister, "Separating and Representing Navigation Concerns in Web Applications". *Lehigh University, Technical Reports*, 2004

[19] Httpunit. http://httpunit.sourceforge.net

[20] Hyperj. http://www.research.ibm.com/hyperspace

[21] T. Isakowitz, E. A. Stohr, and P. Balasubranian. "RMM: A Methodology for Structured Hypermedia Design." *Communications of the ACM*, August 1995

[22] C. Kallepalli and J. Tian. "Measuring and Modeling Usage and Reliability for Statistical Web Testing." *Ieee Transactions on Software Engineering*, November 2001

[23] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, "Aspect-Oriented Programming". *11th Europeen Conf. Object-Oriented Programming*, Springer Verlag. 1997.

[24] D. C. Kung, P. Hsia, and J. Gao. "Testing Object-Oriented." *Software. Wiley-IEEE Press*, 2002

[25] Mercury interactive. http://www.merc–int.com

[26] G. Murphy, A. Lai, R. Walker, and M. Robillard,. "Separating Features in Source Code: An Exploratory Study". *23rd International Conference on Software Engineering*, Toronto, Canada. May, 2001

[27] N. Noda and T. Kishi, "On Aspect-Oriented Design Applying Multi-Dimensional Separation of Concerns on Designing Quality Attributes". *First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems (OOPSLA'99)*, November 1999

[28] P. Peixoto, K. Fung, and D. Lowe. "A Framework for the Simulation of Web Applications." *Fourth International Conference on Web Engineering (ICWE 2004)*, M¨unchen, Germany. July 2004

[29] B. Pekilis. "Multi-Dimensional Separation of Concerns and IBM Hyper/J." *Technical Research Report*, University of Waterloo, Canada. January 2002

[30] Rational Rose Web Modeler, http://www.rational.com

[31] A. Reina, J. Torres, and M. Toro, "Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development". *Workshop of nalysis of Aspect-Oriented Software (AAOS 2003)*, University of Darmstadt, Germany. July 2003

[32] F. Ricca and P. Tonella, "Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions". *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'200)*, Genova, Italy. April 2001

[33] D. Schwabe, R. Pontes, and I. Moura. "OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW." *SigWEB Newsletter*, 8, June 1999

[34] M. Siff and T. Reps, "Identifying modules via concept analysis." In M. J. Harrold and G. Visaggio, editors, Proc. IEEE Intl. Conf. on Software Maintenance, Bari, Italy, 1997. IEEE Comp. Soc. Press.

[35] J. Stanley and M. Sutton "Multiple Dimensions of Concern in Software Testing". *First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems (OOPSLA'99)*, November 1999

[36] P. Tarr, H. Ossher, W. Harrison, J. Stanley, and M. Sutton, "N-degrees of separation: Multi-Dimensional Separation of Concerns". *21st International Conference on SoftwareEngineering*, IEEE Computer Society Press, 1999

[37] P. Tonella and M. Ceccato, "Aspect Mining through the Formal Concept Analysis of Execution Traces". *11th IEEE Working Conference on Reverse Engineering (WCRE 04)*, Netherlands. November 2004

[38] ToscanaJ, http://toscanaj.sourceforge.net