PATTERNS 2010

The Second International Conferences on Pervasive Patterns and Applications

November 21-26, 2010 - Lisbon, Portugal

ComputationWorld 2010 Editors

Ali Beklen, IBM Turkey, Turkey
Jorge Ejarque, Barcelona Supercomputing Center, Spain
Wolfgang Gentzsch, EU Project DEISA, Board of Directors of OGF, Germany
Teemu Kanstren, VTT, Finland
Arne Koschel, Fachhochschule Hannover, Germany
Yong Woo Lee, University of Seoul, Korea
Li Li, Avaya Labs Research - Basking Ridge, USA
Michal Zemlicka, Charles University - Prague, Czech Republic

PATTERNS 2010

Foreword

The Second International Conferences on Pervasive Patterns and Applications [PATTERNS 2010], held between November 21 and 26 in Lisbon, Portugal, targeted the application of advanced patterns, at-large. In addition to support for patterns and pattern processing, special categories of patterns covering ubiquity, software, security, communications, discovery and decision were considered. As a special target, the domain-oriented patterns cover a variety of areas, from investing, dietary, forecast, to forensic and emotions. It is believed that patterns play an important role on cognition, automation, and service computation and orchestration areas. Antipatterns come as a normal output as needed lessons learned.

We take here the opportunity to warmly thank all the members of the PATTERNS 2010 Technical Program Committee, as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to PATTERNS 2010. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the PATTERNS 2010 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that PATTERNS 2010 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the areas pervasive patterns and applications.

We are convinced that the participants found the event useful and communications very open. We also hope the attendees enjoyed the beautiful surroundings of Lisbon, Portugal.

PATTERNS 2010 Chairs:

Eva Blomqvist, STLab ISTC-CNR, Italy
Markus Goldstein, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Teemu Kanstren, VTT, Finland
Fritz Laux, Reutlingen University, Germany
Herwig Manaert, University of Antwerp, Belgium
Guenter Neumann, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Juan Pelaez, U.S. Army Research Laboratory, USA
Lei Shu, Osaka University, Japan
Zhenzhen Ye, iBasis, Inc., Burlington, USA
Michal Zemlicka, Charles University - Prague, Czech Republic

PATTERNS 2010

Committee

PATTERNS Advisory Chairs

Academia
Herwig Manaert, University of Antwerp, Belgium
Michal Zemlicka, Charles University - Prague, Czech Republic
Fritz Laux, Reutlingen University, Germany
Lei Shu, Osaka University, Japan

Research Institutes
Teemu Kanstren, VTT, Finland
Guenter Neumann, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Juan Pelaez, U.S. Army Research Laboratory, USA
Eva Blomqvist, STLab ISTC-CNR, Italy
Markus Goldstein, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Zhenzhen Ye, iBasis, Inc., Burlington, USA

PATTERNS 2010 Technical Program Committee

Junia Anacleto, Federal University of Sao Carlos, Brazil
Francesca Arcelli, University of Milano Bicocca, Italy
Li Bai, University of Nottingham, UK
Eva Blomqvist, STLab ISTC-CNR, Italy
Dave Bustard, University of Ulster-Coleraine, UK
Jean-Charles Créput, Universite de Technologie de Belfort Montbeliard (UTBM), France
Angélica de Antonio, Universidad Politécnica de Madrid (UPM), Spain
Alessandra Scotto di Freca, University of Cassino, Italy
Kamil Dimililer, Near East University, Turkey
Petre Dini, Concordia University, Canada / IARIA, USA
Jürgen Ebert, Universität Koblenz-Landau, Germany
Eduardo B. Fernandez, Florida Atlantic University - Boca Raton, USA
Francesco Fontanella, Università degli Studi di Cassino, Italy
Harald Gjermundrod, University of Nicosia, Cyprus
Markus Goldstein, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Carmine Gravino, University of Salerno - Fisciano, Italy
Yann-Gaël Guéhéneuc, École Polytechnique de Montréal, Canada
Mohamed Farouk Abdel Hady, University of Ulm, Germany
Jon Hall, The Open University, UK
Chih-Cheng Hung, Southern Polytechnic State University, USA
Shareeful Islam, Technische Universität München, Germany
Hermann Kaindl, TU-Wien, Austria
Teemu Kanstren, VTT, Finland

Copyright Information

Table of Contents

A Formal Language of Pattern Compositions

Ian Bayley and Hong Zhu
Department of Computing and Electronics, Oxford Brookes University
Oxford OX33 1HX, UK. Email: ibayley@brookes.ac.uk, hzhu@brookes.ac.uk

***Abstract*—In real applications, design patterns are almost always to be found composed with each other. Correct application of patterns therefore relies on precise definition of these compositions. In this paper, we propose a set of operators on patterns that can be used in such definitions. These operators are restriction of a pattern with respect to a constraint, superposition of two patterns, and a number of structural manipulations of the pattern's components. We also report a case study on the pattern compositions suggested informally in the Gang of Four book in order to demonstrate the expressiveness of the operators.**

***Keywords*-Design patterns, Pattern composition, Object oriented design, Formal methods.**

I. Introduction

As codified reusable solutions to recurring design problems, design patterns play an increasingly important role in the development of software systems [1], [2]. In the past few years, many such patterns have been identified, catalogued [1], [2], formally specified [3]–[6], and included in software tools [7]–[9]. Although each pattern is specified separately, they are usually to be found composed with each other in real applications. It is therefore imperative to represent pattern compositions precisely and formally so that the correct usage of composed patterns can be verified and validated.

However, while many approaches to pattern formalisation have been proposed, very few authors have investigated pattern composition formally. In [10], Taibi discussed composition but went no further than illustrating it with an example. In [11], we formally defined a pattern composition operator. It is universal but not very flexible for practical uses. In this paper, we revise the work, taking a radically different approach. Instead of defining a single universal composition operator, we formally define a set of operators, with which each sort of composition can be accurately and precisely expressed.

The remainder of the paper is organised as follows. Section II reviews the different approaches to pattern formalisation to give the background of the paper. Section III formally defines the set of six operators. Section IV gives an example to illustrate how compositions can now be specified. Section V reports a case study in which we used the operators to realise all the pattern combinations suggested by the Gang of Four (GoF) book [1]. Section VI concludes the paper with a discussion of related works and future work.

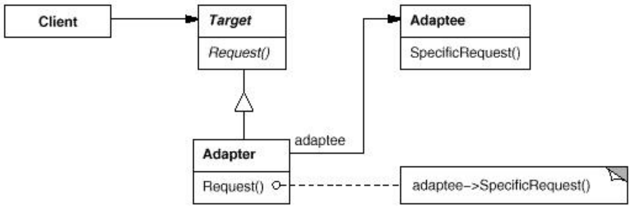
II. Background

In the past few years, researchers have advanced several approaches to the formalisation of design patterns. In spite of the differences in their formalisms, the basic underlying ideas are quite similar. In particular, valid pattern instances are usually specified using statements that constrain their structural features and sometimes their behavioural features too. The structural constraints are typically assertions that certain types of components exist and have a certain static configuration. The behavioural constraints, on the other hand, detail the temporal order of messages exchanged between the components that realise the designs.

The various approaches to pattern formalisation differ in how they represent software systems and in how they formalise the predicate. For example, Eden's predicates are on the source code of object-oriented programs [5] but they are limited to structural features. Taibi's approach in [4] is similar but he takes the further step of adding temporal logic for behavioural features. In contrast, our predicates are built up from primitive predicates on UML class and sequence diagrams [6]. These primitives are induced from GEBNF, which is an extension of BNF for graphical modelling languages [12]. Nevertheless, the operators on design patterns used in this paper are generally applicable and independent of the particular formalism used. Still, the examples used to illustrate the operators and our formalism come from our previous work [6].

As examples, Figures 1 and 2 show the specification of the Object Adapter and Composite design patterns. The class diagrams from the GoF book have been reproduced to enhance readability. The primitive predicates and functions we use are explained in Table I. All of them are either induced directly from the GEBFN definition of UML, or are defined formally in terms of such predicates.

In general, a design pattern P can be defined abstractly as an ordered pair $\langle V, Pr\rangle$, where Pr is a predicate on the domain of some representation of software systems, and V is a set of declarations of variables free in Pr. In other words, Pr specifies the structural and behavioural features of the pattern and V specifies its components. Let $V = \{v_1 : T_1, \cdots, v_n : T_n\}$, where v_i are variables that range over the type T_i of software elements. The semantics of the



Specification 1: (*Object Adapter Pattern*)
Components

1) $Target, Adapter, Adaptee \in classes$,
2) $requests \subseteq Target.opers$,
3) $specreqs \subseteq Adaptee.opers$

Static Conditions

1) $Adapter \mathrel{-\!\!\triangleright}^{+} Target$, $Adapter \longrightarrow^{+} Adaptee$,
2) $CDR(Target)$

Dynamic Conditions

1) $\forall o \in requests \cdot \exists o' \in specreqs \cdot (calls(o, o'))$

Figure 1. Specification of Object Adapter Pattern

Specification 2: (*Composite*)
Components

1) $Component, Composite \in classes$,
2) $Leaves \subseteq classes$,
3) $ops \subseteq Component.opers$

Static Conditions

1) $ops \neq \emptyset$
2) $\forall o \in ops.isAbstract(o)$,
3) $\forall l \in Leaves \cdot (l \mathrel{-\!\!\triangleright}^{+} Component$
   $\wedge \neg(l \diamond\!\!\longrightarrow^{+} Component))$
4) $isInterface(Component)$
5) $Composite \mathrel{-\!\!\triangleright}^{*} Component$
6) $Composite \diamond\!\!\longrightarrow^{+} Component$
7) $CDR(Component)$

Dynamic Conditions

1) any call to $Composite$ causes follow-up calls

$$\forall m \in messages \cdot \exists o \in ops \cdot$$
$$(toClass(m) = Composite \wedge m.sig \approx o \Rightarrow$$
$$\exists m' \in messages \cdot calls(m, m') \wedge m'.sig \approx m.sig)$$

2) any call to a leaf does not

$$\forall m \in messages \cdot \exists o \in ops \cdot$$
$$toClass(m) \in Leaves \wedge m.sig \approx o \Rightarrow$$
$$\neg \exists m' \in messages \ . \ calls(m, m') \wedge m'.sig \approx m.sig)$$

Figure 2. Specification of Composite Pattern

specification is a ground predicate in the form.

$$\exists v_1 : T_1 \cdots \exists v_n : T_n \cdot (Pr) \quad (1)$$

In the sequel, we write $Spec(P)$ to denote the predicate (1) above, $Vars(P)$ for the set of variables declared in V, and $Pred(P)$ for the predicate Pr.

Table I
THE FUNCTIONS AND PREDICATES USED IN THE EXAMPLES

ID	Meaning
$classes$	The set of class nodes in the class diagram
$opers$	The operations contained in the class node
sig	The signature of the message
$X \mathrel{-\!\!\triangleright}^{+} Y$	Class X inherits class Y directly or indirectly
$X \longrightarrow^{+} Y$	There is an association from class X to Y directly or indirectly
$X \diamond\!\!\longrightarrow^{+} Y$	There is an composite or aggregate relation from X to Y directly or indirectly
$isInterface(X)$	Class X is an interface
$CDR(X)$	No messages are sent to a subclass of X from outside directly
$calls(x, y)$	Operation x calls operation y
$isAbstract(op)$	Operation op is abstract
$toClass(m)$	The class that message m is sent to
$X \approx Y$	Operations X and Y share the same name

We can formally define the conformance of a design model m to a pattern P, written as $m \models P$, and reason about the properties of instances based on the patterns they conform to, but we omit the details here for the sake of space. Readers are referred to [6] and [12].

III. OPERATORS ON PATTERNS

We now formally define the operators on design patterns.

A. Restriction operator

The restriction operator was first introduced in our previous work [11], where it is called the *specialisation* operator.

Definition 1: (Restriction operator)
Let P be given pattern and c be a predicate defined on the components of P. A restriction of P with constraint c, written as $P\,[c]$, is the pattern obtained from P by imposing the predicate c as an additional condition on the pattern. Formally,

1) $Vars(P[c]) = Vars(P)$,
2) $Pred(P[c]) = (Pred(P) \wedge c)$. □

For example, a variant of Composite pattern in which there is only one leaf, called $Composite_1$ in the sequel, can be formally defined as follows.

$$Composite_1 = Composite[\#Leaves = 1].$$

Restriction is frequently used in the case study, particularly in the form $P[u = v]$ for pattern P and variables u and v of the same type. This expression denotes the pattern obtained from P by unifying u and v to make them the same element.

The restriction operator does not introduce any new components into the structure of a pattern, but the following operators do.

B. Superposition operator

Definition 2: (Superposition operator)

Let P and Q be two patterns. Assume that the component variables of P and Q are disjoint, i.e., $Vars(P) \cap Vars(Q) = \emptyset$. The *superposition* of P and Q, written $P * Q$, is a pattern that consists of both pattern P and pattern Q as is formally defined as follows.

1) $Vars(P * Q) = Vars(P) \cup Vars(Q)$;
2) $Pred(P * Q) = Pred(P) \wedge Pred(Q)$. □

For example, the superposition of Composite and Adapter patterns, $Composite * Adapter$, requires each instance to contain one part that satisfies the Composite pattern and another that satisfies the Adapter pattern. These parts may or may not overlap, but the following expression does enforce an overlap, requiring that a class in $Leaves$ be the target of an Adapter.

$$(Composite * Adapter)[Target \in Leave]$$

The requirement that $Vars(P)$ and $Vars(Q)$ be disjoint is easy to fulfil using renaming. An appropriate notation for this will be introduced later.

C. Extension operator

Definition 3: (Extension operator)

Let P be a pattern, V be a set of variable declarations that are disjoint with P's component variables (i.e., $Vars(P) \cap V = \emptyset$), and c be a predicate with variables in $Vars(P) \cup V$. The extension of pattern P with components V and linkage condition c, written as $P\#(V \bullet c)$, is defined as follows.

1) $Vars(P\#(V \bullet c)) = Vars(P) \cup V$;
2) $Pred(P\#(V \bullet c)) = Pred(P) \wedge c$. □

D. Flatten operator

Definition 4: (Flatten Operator)

Let P be a pattern, $Vars(P) = \{x : \mathbb{P}(T), x_1 : T_1, \cdots, x_k : T_k\}$ and $Pred(P) = p(x, x_1, \cdots, x_k)$, and $x' \notin Vars(P)$. The flattening of P on variable x, written $P \Downarrow x \backslash x'$, is the pattern that has the following property.

1) $Vars(P \Downarrow x \backslash x') = \{x' : T, x_1 : T_1, \cdots, x_k : T_k\}$;
2) $Pred(P \Downarrow x \backslash x') = p'(x', x_1, \cdots, x_k)$,

where $p'(x', x_1, \cdots, x_k) = p(\{x'\}, x_1, \cdots, x_k)$. That is, the predicate p' is obtained by replacing all free occurrences of variable x with expression $\{x'\}$. □

Note that, $\mathbb{P}(T)$ denotes the power set of T. For example, in the specification of Composite pattern, the component variable $Leaves \subseteq classes$ is a subset of classes. Its type is $\mathbb{P}(classes)$.

For example, the single-leaf variant of Composite pattern $Composite_1$ can also be defined as follows.

$$Composite_1 = Composite \Downarrow Leaves \backslash Leaf$$

As an immediate consequence of this definition, we have the following property. For $x_1 \neq x_2$ and $x'_1 \neq x'_2$,

$$(P \Downarrow x_1 \backslash x'_1) \Downarrow x_2 \backslash x'_2 = (P \Downarrow x_2 \backslash x'_2) \Downarrow x_1 \backslash x'_1. \quad (2)$$

Therefore, we can overload the $\Downarrow$ operator to a set of component variables. Let X be a subset of P's component variables all of power set type, i.e., $X = \{x_1 : \mathbb{P}(T_1), \cdots, x_n : \mathbb{P}(T_n)\} \subseteq Vars(P), n \geq 1$ and $X' = \{x'_1, \cdots, x'_n\}$ such that $X' \cap Vars(P) = \emptyset$. We write $P \Downarrow X \backslash X'$ to denote $P \Downarrow x_1 \backslash x'_1 \Downarrow \cdots \Downarrow x_n \backslash x'_n$.

Note that our pattern specifications are closed formulae, containing no free variables. Although the names given to component variables greatly improve readability, they have no effect on semantics so, in the sequel, we will often omit new variable names and write simply $P \Downarrow x$ to represent $P \Downarrow x \backslash x'$.

E. Generalisation operator

Definition 5: (Generalisation operator)

Let P be a pattern, $x \in Vars(P) = \{x : T, x_1 : T_1, \cdots, x_k : T_k\}$. The *generalisation* of P on variable x, written $P \Uparrow x \backslash x'$, is defined as follows.

1) $Vars(P \Uparrow x \backslash x') = \{x' : \mathbb{P}(T), x_1 : T_1, \cdots, x_k : T_k\}$,
2) $Pred(P \Uparrow x \backslash x') = \forall x \in x' \cdot Pred(P)$.□

For example, we can define the Composite pattern as a generalisation of the single-leaf variant $Composite_1$, i.e.,

$$Composite = Composite_1 \Uparrow Leaf \backslash Leaves$$

We will use the same syntactic sugar for $\Uparrow$ as we do for $\Downarrow$. We will often omit the new variable name and write $P \Uparrow x$. Thanks to an analogue of Equation 2, we can and also will promote the operator $\Uparrow$ to sets.

F. Lift operator

The lift operator was first introduced in our previous work [11]. This time, we decompose the definition of a pattern slightly differently, into the existentially quantified class components $CVars(P)$ and the remainder of the predicate $OPred(P)$, which includes the declarations of the operations, existentially quantified at the outermost. Then we can define lifting as follows.

Definition 6: (Lift Operator)

Let P be a pattern and $CVars(P) = \{x_1 : T_1, \cdots, x_n : T_n\}$, $n > 0$ and $OPred(P) = p(x_1, \cdots, x_n)$. Let $X = \{x_1, \cdots, x_k\}$, $1 \leq k < n$, be a subset of the variables in the pattern. The lifting of P with X as the key, written $P \uparrow X$, is the pattern defined as follows.

1) $CVars(P \uparrow X) = \{xs_1 : \mathbb{P}T_1, \cdots, xs_n : \mathbb{P}T_n\}$,
2) $OPred(P \uparrow X) = \forall x_1 \in xs_1 \cdots \forall x_k \in xs_k \cdot \exists x_{k+1} \in xs_{k+1} \cdots \exists x_n \in xs_n \cdot p(x_1, \cdots, x_n)$. □

Where the key set is singleton, we omit the set brackets for simplicity, so we write $P \uparrow x$ instead of $P \uparrow \{x\}$.

For example, Figure 3 is the pattern defined by expression $Adapter \uparrow Target$.

Informally, lifting a pattern P results in a pattern P' that contains a number of instances of pattern P. For example, $Adapter \uparrow Target$ is the pattern that contains a number of $Target$s of adapted classes. Each of these has a dependent $Adapter$ and $Adaptee$ class configured as in the original $Adapter$ pattern. In other words, the component $Target$ in the lifted pattern plays a role similar to the *primary key* in a relational database.

Specification 3: (*Lifted Object Adapters Pattern*)
Components

1) $Targets, Adapters, Adaptees \subseteq classes,$

Conditions

1) $\forall Adaptee \in Adaptees \cdot \exists specreqs \in Adaptee.opers,$
2) $\forall Target \in Targets \cdot \exists requests \in Target.opers,$
3) $\forall Target \in Targets \cdot CDR(Target),$
4) $\forall Target \in Targets \cdot$
   $\exists Adapter \in Adapters, Adaptee \in Adaptees \cdot$
   a) $Adapter \longrightarrow\!\!\!\triangleright Target,$
   b) $Adapter \longrightarrow Adaptee,$
   c) $\forall o \in Target.requests \cdot$
      $\exists o' \in Adaptee.specreqs \cdot (calls(o, o')))$

Figure 3. Specification of Lifted Object Adapter Pattern

IV. Example

The composition of patterns is often represented graphically with *Pattern:Role* annotations [13]. An example is Figure 4, taken from [13] (p131). It is composed from five patterns: *Command*, *Command Processor*, *Strategy*, *Composite*, and *Memento*. The composition can be easily expressed as an expression in the operators of this paper.

First though, we must introduce a notation for renaming the variables in one pattern to make them disjoint from those in another. Let $x \in Vars(P)$ be a component of pattern P and $x' \notin Vars(P)$. The systematic renaming of x to x' is written as $P[x' := x]$. Obviously, the renaming does not affect model satisfiability. (Formally, for all models m, we have $m \models P \Leftrightarrow m \models P[x' := x]$.) Let $P[v := x = y]$ be syntactic sugar for $P[x = y][v := x][v := y]$, i.e., both x and y are renamed and equated to v. Similarly, let $P[v := x \in y]$ abbreviate $P[x \in y][v := x]$.

Then we can translate each annotation group as a single restriction, representing the diagram with the following expression:

$$\begin{aligned}
&(Command * CommandProcessor * Strategy * \\
&\quad Composite * Memento) \\
&[commandProcessor := context] \\
&[command := CPcommand = component \in caretakers] \\
&[(composite \in caretakers) \\
&\quad \wedge (composite \in concreteCommands)] \\
&[concreteCommand := composite] \\
&[concreteCommands := leaf \in caretakers] \\
&[Logging := strategy] \\
&[ConcreteLoggingStrategies := concreteStrategies]
\end{aligned}$$

Note that compositions such as this, representable in a graphical form with annotations, can always be represented using the restriction and superposition operators, but not all the examples in the next section, so the graphical notation is not expressive enough, nor are the two operators when used on their own.

V. Case Study

In the GoF book, the documentation for each pattern concludes with a brief section entitled Related Patterns. As the title suggests, it compares and contrasts patterns, but more importantly, it makes suggestions for how other patterns may be used with the one under discussion. These suggestions are summarised in a diagram on the back cover. Our case study is to formalise them all as expressions with the operators from this paper and predicates specifying the patterns; the latter can be found in [6]. For example, on page 106 of the GoF book, it is stated that "*A Composite is what the builder often builds*". This can be formally specified as follows.

$$(Builder * Composite)[Product = Component].$$

Figure 5 shows our coverage of these relationships, based on the aforementioned GoF diagram, with each numbered relationship summarised in the corresponding row of Table II. $Composite_1$ is as defined in Section III (any of the equivalent definitions can be used) and the notation $P.x$ denotes the variable x in pattern P.

Five new arrows have been added to the diagram and numbered in bold font. These relationships were discussed in GoF but omitted from its version of diagram. We were still able to formalise them because GoF contained the information we needed to do so. On the other hand, four arrows from the original diagram have been kept but labelled with asterisks in place of numbers. These are the relationships that do not represent compositions and thus could not be formalised as expressions. In particular, it is a specialisation relation that links Composite and Interpreter, which can be formally proved; see [14]. The relationship between Decorator and Strategy is a comparison of the two, not a composition suggestion, so is the relationship between Strategy and Template Method. That between Iterator and Visitor, on the other hand, has not been formalised for the different reason that it is mentioned in GoF only on the diagram, and not expanded upon in the main text.

The case study has demonstrated that the operators defined in this paper are expressive to define compositions of design patterns.

VI. Conclusion

In this paper, we proposed a set of operators on design patterns that enable compositions to be formally defined with flexibility. We illustrated the operators with examples. We also reported a case study of the relationships between design patterns suggested by GoF [1]. It demonstrated the expressiveness of the operators in the composition of patterns.

Formal reasoning about both design patterns and their compositions can be naturally supported by formal deduction in first-order logic. This activity is well understood, and well supported by software tools such as theorem provers. In the case study, we have noticed that some pattern compositions can be represented in different but equivalent expressions. For example, we have seen in Section III that $Composite_1$

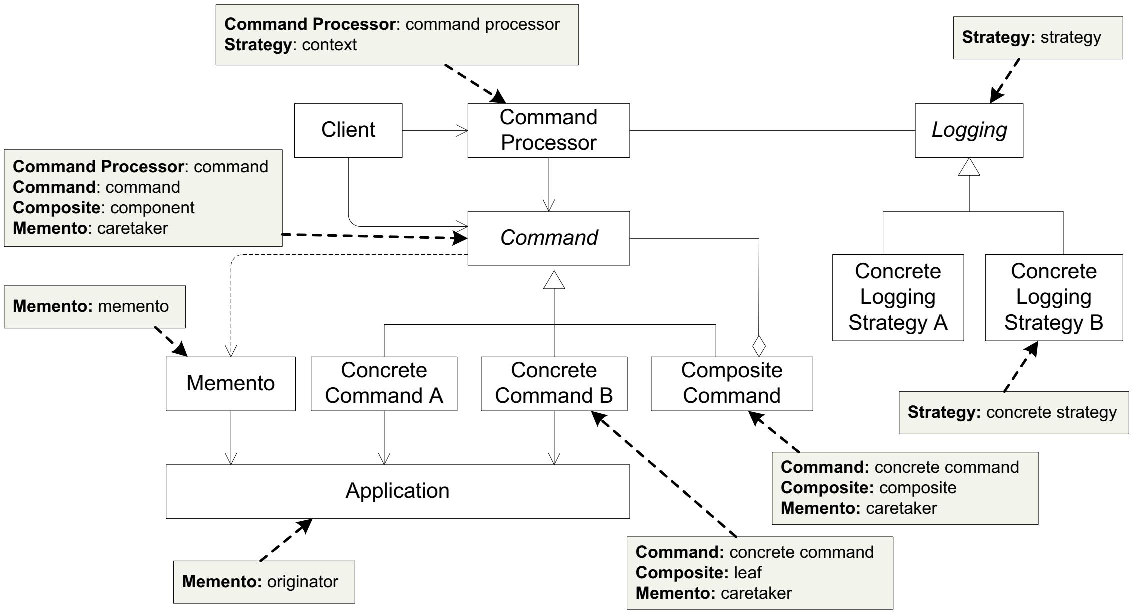


Figure 4. Example of pattern composition represented in the form of *Pattern:Role* annotation

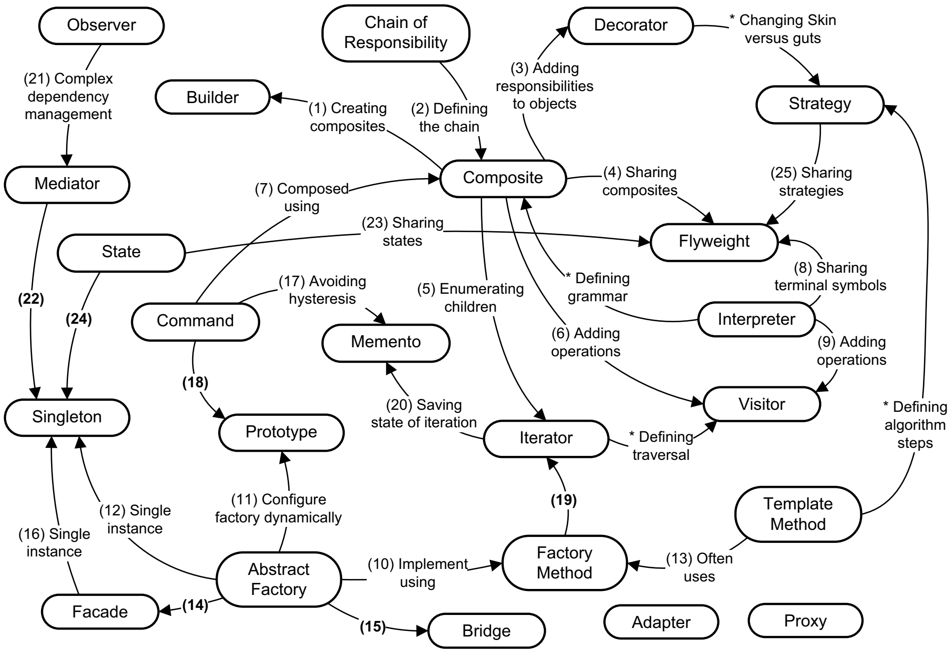


Figure 5. Case Study on Formalising Relationships between GoF Patterns

Table II
FORMAL DEFINITIONS OF THE COMPOSITIONAL RELATIONSHIPS BETWEEN PATTERNS

No.	Definition of the compositional relationship
1	$(Builder * Composite)[Product = Component]$
2	$(Composite * ChainOfResponsibility)[Handler = Component \wedge Operation = Handle \wedge multiplicity = 1]$
3	$(Composite_1 * Decorator)[Composite_1.Component = Decorator.Component \wedge$ $Composite_1.Operation = Decorator.Operation \wedge ConcreteComponent = Leaf \wedge Decorator = Composite_1]$
4	$(Composite * Flyweight)[Leafs = \{ConcreteFlyweight, UnsharedConcreteFlyweight\}]$
5	$(Composite * Iterator')[ConcreteAggregate = Component]$
6	$(Composite * Visitor)[Element = Component \wedge Operation = Accept(v) \wedge ConcreteElements = \{Leaf, Composite\}]$
7	$(Composite * Command)[Command = Component \wedge execute = operation \wedge ConcreteCommand = Leaf]$
8	$(Interpreter * Flyweight)[TerminalExpression = Flyweight]$
9	$(Interpreter * Visitor)[Element = AbstractExpression \wedge Interpret = Accept(v) \wedge$ $ConcreteElements = \{NonTerminalExpression, TerminalExpression\}]$
10	$(AbstractFactory * ((FactoryMethod \uparrow Product) \Uparrow FactoryMethod))[Creator = AbstractFactory \wedge$ $\#AnOperations = 1 \wedge createMethods \subseteq FactoryMethods \wedge ConcreteCreators = ConcreteFactories \wedge$ $Products = AbstractProducts \wedge AbstractFactory.ConcreteProducts = FactoryMethod.ConcreteProducts]$
11	$(AbstractFactory * (Prototype \uparrow Client))[ConcreteFactories \subseteq Clients \wedge$ $CreateProductOperations \subseteq Operations \wedge AbstractProducts \subseteq Prototypes]$
12	$(AbstractFactory * (Singleton \uparrow \{Singleton\}))[Singletons \subseteq ConcreteFactories]$
13	$(TemplateMethod * FactoryMethod)[AbstractClass = Creator \wedge TemplateMethod = AnOperation]$
14	$(AbstractFactory * Facade)[AbstractFactory = Facade]$
15	$(AbstractFactory * Bridge)[AbstractProducts = \{Abstraction, Implementor\}]$
16	$(Facade * Singleton)[Facade = Singleton]$
17	$(Command * Memento)[Originator = Command]$
18	$(Command * Prototype)[Command = Prototype]$
19	$(Iterator * FactoryMethod)[Creator = Aggregate \wedge Product = Iterator \wedge ConcreteCreator = ConcreteAggregate \wedge$ $ConcreteProduct = ConcreteIterator \wedge AnOperation = CreateIterator]$
20	$(Memento * Iterator)[ConcreteAggregate = Originator]$
21	$(Mediator * Observer)[ConcreteColleagues = \{ConcreteSubject, ConcreteObserver\}]$
22	$(Mediator * Singleton)[ConcreteMediator = Singleton]$
23	$(Flyweight * State)[Flyweight = State \wedge Handle = Operation(extrinsicState)]$
24	$(State * (Singleton \Uparrow Singleton))[Singletons \subseteq ConcreteStates]$
25	$(Strategy * Flyweight)[Strategy = Flyweight \wedge algorithmInterface = Operation(extrinsicState)]$

can be expressed either using the restriction operator or using the flatten operator, and these two expressions are equivalent. We are now investigating the algebraic laws that the operators obey. This will lead us to a calculus of pattern composition to enable us to reason about the equivalence of such expressions.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[2] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, 2nd ed. Prentice Hall, 2003.

[3] T. Mikkonen, "Formalizing design patterns," in *Proc. of ICSE'98*. IEEE CS, April 1998, pp. 115–124.

[4] T. Taibi, D. Check, and L. Ngo, "Formal specification of design patterns-a balanced approach," *Journal of Object Technology*, vol. 2, no. 4, July-August 2003.

[5] E. Gasparis, A. H. Eden, J. Nicholson, and R. Kazman, "The design navigator: charting Java programs," in *Proc. of ICSE'08, Companion Volume*, 2008, pp. 945–946.

[6] I. Bayley and H. Zhu, "Formal specification of the variants and behavioural features of design patterns," *Journal of Systems and Software*, vol. 83, no. 2, pp. 209–221, Feb. 2010.

[7] D. Hou and H. J. Hoover, "Using SCL to specify and check design intent in source code," *IEEE TSE*, vol. 32, no. 6, pp. 404–423, June 2006.

[8] N. Nija Shi and R. Olsson, "Reverse engineering of design patterns from JAVA source code," in *Proc. of ASE'06*, Sept. 2006, pp. 123–134.

[9] D. Mapelsden, J. Hosking, and J. Grundy, "Design pattern modelling and instantiation using dpml," in *Proc. of CRPIT '02*. Australian Computer Society, Inc., 2002, pp. 3–11.

[10] T. Taibi, "Formalising design patterns composition," *Software, IEE Proceedings*, vol. 153, no. 3, pp. 126–153, June 2006.

[11] I. Bayley and H. Zhu, "On the composition of design patterns," in *Proc. of QSIC'08*, IEEE CS, pp. 27–36.

[12] H. Zhu, "On the theoretical foundation of meta-modelling in graphically extended bnf and first order logic," in *Proc. of TASE 2010*. IEEE CS, Aug. 2010, (in press), Available online at http://cms.brookes.ac.uk/staff/HongZhu/Publications /TASE2010.pdf

[13] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. John Wiley & Sons Ltd., 2007, vol. 5.

[14] I. Bayley and H. Zhu, "Formalising design patterns in predicate logic," in *Proc. of SEFM'07*. IEEE CS, Sept. 2007, pp. 25–36.

PLOP: A Pattern for Learning Objects for Programming

Luis A. Álvarez-González, Valeria Henríquez N., Erick Araya and Fabiola Cárdenas
Grupo de Investigación en Tecnología de Aprendizaje
Universidad Austral de Chile, Valdivia, Chile
lalvarez@inf.uach.cl, valeria.a.henriquez@gmail.com, earaya@uach.cl, fabiolacardenas@uach.cl

***Abstract*—This paper shows a Pattern of Interaction to enhance the construction process of learning objects for introductory programming languages. All the phases of the design of the pattern are showed. In the phase of educational design, some elements of software engineering are included. In the phase of multimedia design, the information areas are showed and also the sub-areas for each information area. In the implementation phase, the class diagrams of the pattern are showed. Finally, the evaluation and conclusions are presented.**

Keywords-components; learning objects; interacction patterns; programming languages.

I. Introduction

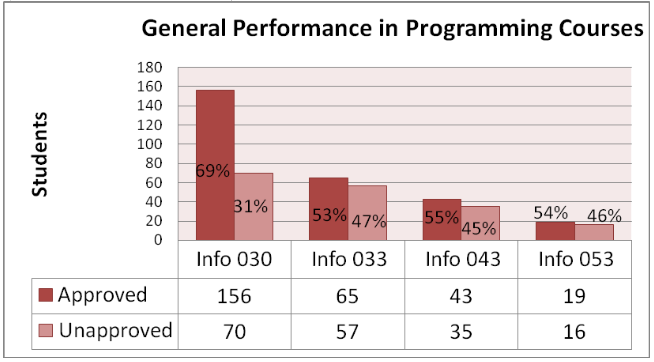
At the Universidad Austral de Chile, located in Valdivia, south of Chile, it is important to incorporate new teaching methods to enhance learning achievement of their students, especially in introductory programming courses where it has been detected a low level of approval[1], 69% the first, 53% second, the third 55% and 54% in the fourth semester (courses Info030, Info033, Info043 and Info053 in Figure 1). The data presented clearly demonstrates the need to develop educational resources to support teaching, especially for struggling students during their first steps in programming, as well as support teachers in the production process of educational material.

Three projects focused in building learning objects (hereafter LOs) for programming languages were found as the most important:

- The CodeWitz Project which aims to plan, produce and evaluate illustrations, animations and visual support for programming students and lecturers [4]. In the CodeWitz network are participating 29 universities in 21 countries, only the universities of the network can use the CodeWitz LOs.
- The Project "A Programming System Education Based on Program Animation", Gakugei University, Tokyo, Japan, which aims to support students with problems and help them understand the performance of different algorithms, with particular emphasis on implementation and changes occurring in the data structures in memory [6].
- Learning Objects for Introductory Programming Project, developed at London Metropolitan University by the staff in the Learning Technology Research Institute (LTRI) and the Department for Computing, Communications Technology & Mathematics, and at Bolton Institute [1].

However, the LOs of these three projects are not available to any user, neither have a good interaction, moreover they do not have areas to define problems, diagrams or metadata, they are written in English and to build a new one the user must modify an old one. In other words, there is not a tool to build LO.

Then, how the lecturers can be converted into producers of LOs? To answer this question we developed a pattern of interaction and a methodology that allows the creation of new LOs following a repeatable process that can be used by any lecturers. With these LOs, it is intended that students can follow the execution of program at a code level, step by step in an interactive fashion to be able to learn for themselves by promoting metacognition, autonomy and respecting the learning pace, regardless of their location (available on Internet).



	Info 030	Info 033	Info 043	Info 053
🟥 Approved	156	65	43	19
▨ Unapproved	70	57	35	16

Figure 1. Approval rate in introductory programming courses.

II. A Pattern of Interaction

A Patterns of Interaction is understood as an effective solution to a recurring problem, because it promotes the reuse of good design, shortens development time and captures the experience of expert designers and programmers in the development of usable interfaces. They

[1] Information given by the School of Informatics Engineering of Universidad Austral de Chile, in the period 2006-2008.

condense the experience into a series of guidelines or recommendations, which can be used by novice developers to the purpose of acquire the ability to design users interfaces [3]. To describe the pattern of interaction, the definition of Welie [8] is used and it is shown in Table 1.

TABLE 1. DESCRIPTION OF THE PATTERNS OF INTERACTION.

Item	**Description**
Name	Represent the execution of an algorithm
Author	Valeria Henríquez-Norambuena
Problem	How to simulate the execution of an algorithm?
Usability Principle	Reduce cognitive charge
Context	This pattern can be used in all educative situations, where you want to understand a running algorithm
Forces	Representation like debug is easy to understand for students
Solutions	Show diagram, memory, I/O and help areas, which can represent the algorithm execution.
Consequences	Develop LOs using this pattern, allows to create a mental image about what happens into the computer when the code is run.

The design of the pattern is based on the methodology of the University of Guadalajara [2].The stages of this proposal are: Educational Design, Interaction Design, Functional Design, Multimedia Design, Implementation and Phase Labeling and Packaging.

A. *Educational Design*

The educational design is based on the methodology developed by the LO Group of the Universidad Autónoma de Aguascalientes [7]. Lecturers are the creators and developers of LOs, assistants and students can play the role of developers, both actors with algorithmic knowledge and training. Like any methodology that uses patterns, you need a bank of patterns for your application. It defines the stages and in each of the activities, artifacts and actors involved. The process includes four phases: Analysis, Development, Testing and Implementation. Figure 2 shows in detail this methodology.

1) *Analysis Phase.* At this stage, it is important to identify the competences we want to develop. Based on this, the author of the content (lecturer), describes in the analysis document, what are the general requirements of the LO and obtains the necessary materials. It is essential that the requirements document is sufficiently clear so that any other parties can continue the process of creating the LO.

a) Activities: To analyze the problem and to obtain the necessary materials.

b) Artifacts: Requirements document.

c) Actors: Author (lecturers).

2) *Development Phase.* The developer selects a pattern that meets the needs identified in the Requirements Specification artifact, generated in the previous phase. After selecting an appropriate standard, we make use of it. This activity is carried out as joint work between the author and coach. The pattern is "filled" with the material made by the author, but contextualisation is provided by the developer. The collaborative work between the actors (author and developer) through regular meetings is likely to decrease the time of the evaluation stage.

a) Activities: To select and to use the pattern.

b) Artifacts: LO and Metadata.

c) Actors: Author (lecturer), developer (programmer).

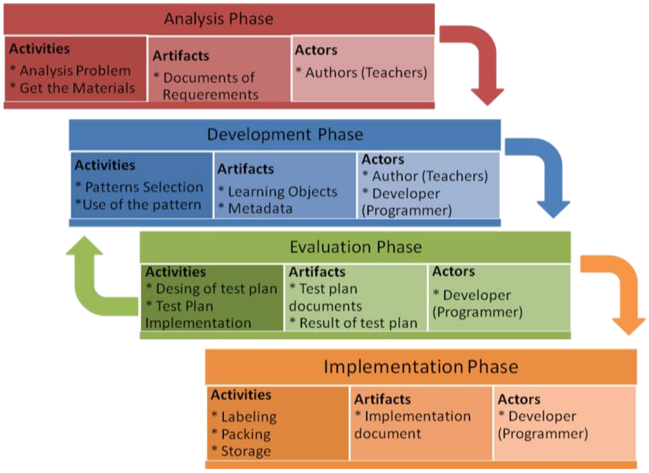


Figure 2. Educational Design Diagram.

3) *Evaluation Phase.* This phase assesses the proper functioning of the LO, i.e. the relevance of content and design. For this phase a plan which specifies the test cases and the results expected is produced. On error, the LO will return to the stage of development to correct the problems identified.

a) Activities: To design and to implement a test plan.

b) Artifacts: Test plan document, results of the test plan report.

c) Actors: Evaluator (lecturer).

4) *Implantation Phase.* After the evaluation phase, the LO is labeled, packaged and stored. At this stage it is important to define the standard to be achieved in the packaging of LO. Then stored in a repository of LO, from which it can be downloaded and/or performed for the interaction and subsequent use.

a) Activities: To label, to package and to store the LO.

b) Artifacts: A document specifying details of the implementation of the LO.

c) Actors: Technical (programmer).

B. *Interaction Design*

For the construction of the pattern of LOs it was decided that the areas of pattern information are as follows:

1) *Diagram Area:* Contains the description of the problem and the graphical solution using a diagram.

2) *Code Area:* Contains the solution in a programming language and simulates the execution of the code. This area contains the following subareas:

a) Implementation Area: Solution in a programming language.

b) Execution Area: Simulation of the potential window displayed by the user.

c) Memory Area: Simulation of the process that occurs in the computer's memory.

d) Explanation Area: Explanation of each code instruction.

e) Control Area: Container buttons.

3) Metadata Area: Contains information related to the LO.

C. Overview of Pattern Operation

- The program will be run step by step through the code area, being the student who decides when to execute the following line of code.
- The program can run automatically and be interrupted if the student decides so.
- The rewind of the code execution is allowed.
- The line of code will be clearly identified.
- Each instruction is synchronized with the support area that will display an explanation for the line running.
- What it is displayed in the memory and execution areas will depend on the line of code
- A LO developed using the interaction pattern must cover any basic problem of programming, in any programming language.

D. Functional Design

To use the pattern, the Adobe Flash CS4 is required. The implementation requires a standard Web browser with Adobe Flash Player, which is available in most browsers. Because the files have `.swf` extension, a LO can be executed in any video player.

E. Multimedia Design

For multimedia design of the different components of the pattern, a model is created. To locate the components, is considered the usual reasoning to read, that is, from left to right and from top to bottom. The model contains two sections of display: the first one, navigation section, which contains three areas ordered by importance to the student: diagram area, code area and metadata area. The second one, located under the navigation section, shows the information for each area (Diagram, Code and Metadata). Figure 3 shows the reasoning in the design.

Figure 4 shows the Diagram Area, which consists of two parts: the left section, an area which describes the problem to be solved and the right section, an area that contains an image with a flow diagram to solve the problem. The order shown allows the student to read the description of the problem and then imagine a solution in a diagram on the right side.

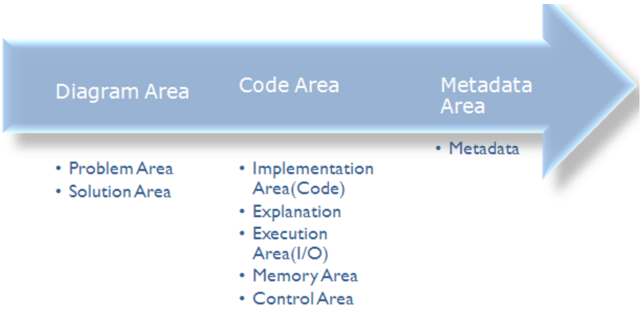


Figure 3. Sections Navigations of the pattern.

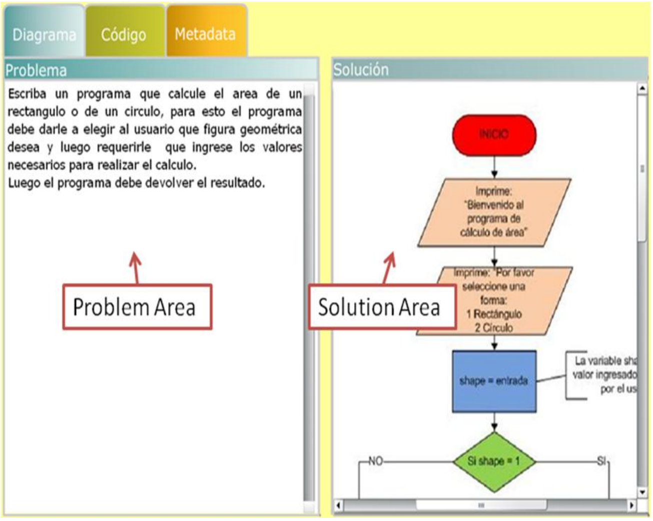


Diagrama
Código
Metadata
Problema
Solución
Problem Area
Solution Area
INICIO
Imprime: "Bienvenido al programa de cálculo de área"
Imprime: "Por favor seleccione una forma: 1 Rectángulo 2 Circulo
shape = entrada
La variable sha valor ingresado por el us
Si shape = 1
NO
SI

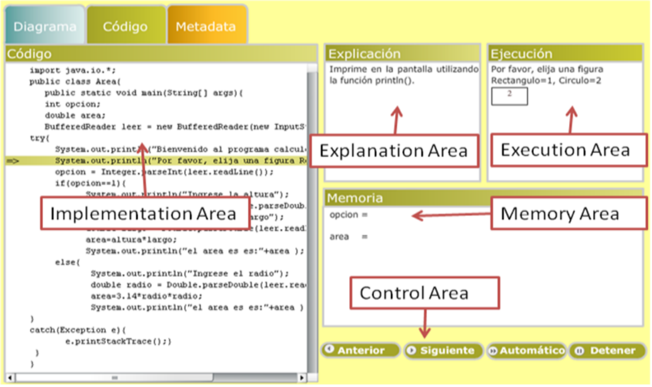
Escriba un programa que calcule el area de un rectangulo o de un circulo, para esto el programa debe darle a elegir al usuario que figura geométrica desea y luego requerirle que ingrese los valores necesarios para realizar el calculo.
Luego el programa debe devolver el resultado.

Figure 4. Diagram area.

Figure 5 shows the Code Area that contains other five areas described in order of importance:

- Implementation Area, which contains text that represents the solution in a programming language.
- Explanation Area contains text synchronized with the Implementation Area that displays information explaining the statement being executed in the code.
- Execution Area. It may contain text or images that simulate the inputs and outputs of the algorithm. This area is the third in importance, because it gives the student an idea of what should happen on the screen of the computer when the code is run.
- Memory Area. Contains animations or images that simulate the processes occurring in the computer memory when the code is run. These changes are not obvious to the student, therefore, lies in fourth place of importance.
- Control Area. Contains buttons that control the implementation of LO.

Figure 6 shows the area of metadata. This area contains a table with information related to the LO. For the metadata was considered a subset of the fields suggested by the standard committees of the IEEE Learning Technologies [5].



```
   import java.io.*;
   public class Area{
      public static void main(String[] args){
      int opcion;
      double area;
      BufferedReader leer = new BufferedReader(new InputS
   try{
        System.out.printl ("Bienvenido al programa calcul
=>      System.out.printl ("Por favor, elija una figura R
        opcion = Integer. arseInt(leer.readLine());
        if(opcion==1){
              System.out. rintln("Ingrese la altura");
                                  .parseDoub
                                  argo");
                                  (leer.read
              area=altura*largo;
              System.out.println("el area es es:"+area );
        else{
               System.out.println("Ingrese el radio");
               double radio = Double.parseDouble(leer.rea
               area=3.14*radio*radio;
               System.out.println("el area es es:"+area )
   }
   catch(Exception e){
          e.printStackTrace();}
    }
   }
```

Imprime en la pantalla utilizando la función println().

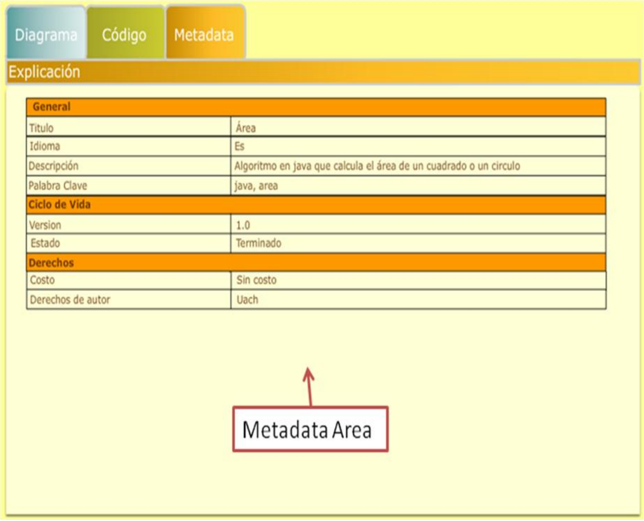
Por favor, elija una figura Rectangulo=1, Circulo=2

2

opcion =

area =

Figure 5. Code area.



General	
Titulo	Área
Idioma	Es
Descripción	Algoritmo en java que calcula el área de un cuadrado o un circulo
Palabra Clave	java, area
Ciclo de Vida	
Version	1.0
Estado	Terminado
Derechos	
Costo	Sin costo
Derechos de autor	Uach

Figure 6. Metadata area.

F. *Implementation*

After the previous stages a LO is built, using Adobe Flash CS4 tools and the ActionScript 3 programing language. For the correct operation of the pattern three files are necessaries (see Figure 7): one with a `.fla` extension, which contains the pattern design, and two files with `.as` extension: **`Patron.as`** that handles the functionality of the standard and **`Mensaje.as`** controls the display of error messages (for example data entry is required and the user does not enter any value).

G. *Labelling and Packaging*

For the development of the pattern of interaction the labeling and packaging process was modified to the creation of a file with `.rar` extension, that contains the pattern and also four LO samples (`HelloWorld`, `HolaNombre`, `Addition` and `Selection`). The sample LOs were built during a validation workshop.

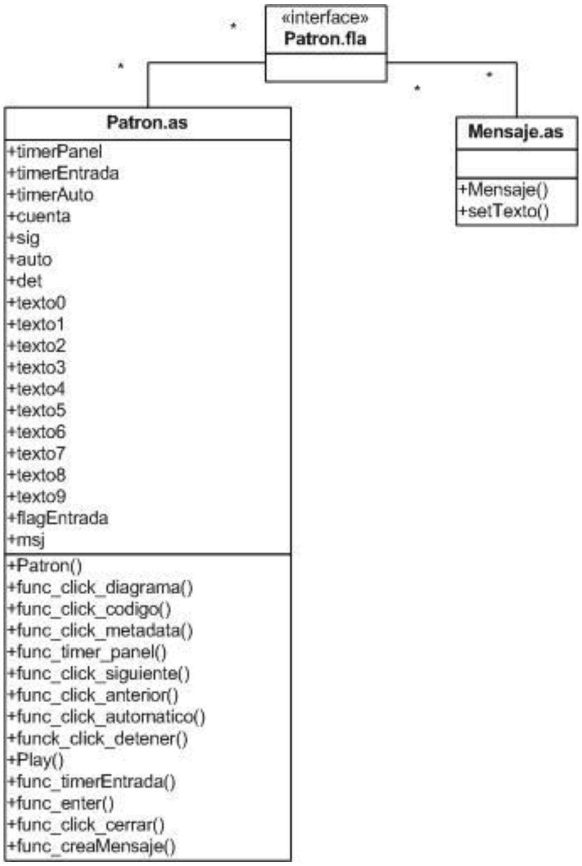


Figure 7. Pattern Class Diagram.

III. Validation of the Methodology and Pattern of Interaction

The validation of the proposed methodology and the pattern of interaction were done through the completion of two workshops and involved a group of six lecturers and four assistant students. Most lecturers give classes in programming at the same university.

A. *Validation Workshop*

The validation workshop was practical and considered three stages.

- Exposure of the general aspects, of the motivations and descriptions of the interaction pattern, including the educational design.
- Practical activities. Participant lecturers took the role of each actor, i.e., educational designer, technical developer and students following a LO. All these activities were conducted in order to gain a better understanding of the methodology (educational design) and the pattern.
- Instrument validation. Implementation of an online survey to participants of the workshop was applied. The survey form contains two sections:

The first seeks to validate the three criteria to be considered when using the pattern of interaction, that is, areas of information, multimedia content and the interaction of LO with the student.

The second refers to the pedagogical design, which seeks to validate the possible impact of using it in the development of LO.

This was aimed at validating this project as a solution to the identified needs.

B. *Validation Results*

Considering nine responses of ten participants in the workshops and a score from 1 to 5, according to Likert's scale (http://es.wikipedia.org/wiki/Escalas_Likert), being 1 strongly disagree and 5 strongly agree. The survey results of the interaction pattern can be seen in Table 2., the average assessment was 4.2 points. It should be noted that the question N° 2 related to sub-areas of information, look better evaluated with 4.7 points.

TABLE 2. SURVEY RESULTS OF VALIDATION PATTERN

Nº	Questions	Score
1	The three areas of information (Diagram, Code and Metadata) seem sufficient for learning a basic algorithmic problem?	4,4
2	The subareas (Problem, Solution Diagram, Solution, Explanation, Implementation, Memory and Control) seem sufficient for learning a basic algorithmic problem?	4,7
3	Do you think the use of the PLOP interaction pattern for the creation of learning objects may help improve the performance of struggling students in introductory programming courses?	4,0
4	The interaction with the areas and subareas are natural for you?	4,0
5	Do you think that the images, text and buttons used are appropriate?	4,4
6	Do you think the use of the pattern of interaction could be enhance in terms of productivity, the creation of learning objects for introductory programming?	3,9
	Average	**4,2**

The survey results for the educational design are shown in Table 3; the average evaluation was 4.05 points.

TABLE 3. SURVEY RESULTS OF EDUCATIONAL DESIGN

Nº	Questions	Score
1	Do you think that the phases of the methodology are appropriate to build Learning Objects?	4,0
2	Do you think that the use of this methodology could improve productivity in developing Learning Objects?	4,1
	Average	**4,05**

Note that this methodology cannot be compared with other similar projects [1][4][6], since there are no known existing projects on patterns applied to learning objects for programming teaching. In the Codewitz Project [4] most learning objects are built using the Macromedia Director (multimedia application authoring platform). To build another one, usually a previous learning object is used. For the other projects [1][6] there is not available information about the building. In other words, none of the previous projects use patterns.

IV. CONCLUSIONS

Considering that only three projects were found with similar objectives, a pattern of interaction in Spanish is built. This pattern considers significant improvements in the areas of information (content distribution), design (colors, buttons, etc.) and metadata for LO. The pattern is enhanced and adapted from the pattern developed by Delgado, Morales, Gonzalez and Chan. The enhancement is because the methodology proposed by Osorio, Muñoz and Alvarez was improved considering software engineering aspects and is included in the pattern. The adaptation is because the pattern and methodology is developed for programming languages learning objects.

The proposed interaction pattern corresponds to a tool for building LOs to support computer programming, so that lecturers can improve approval ratings. Given this context, lecturers are relatively agreeing (Average score 4,2 in Table 2.) that the pattern can serve to build LOs which to improve their students' learning. However, when asked whether the subareas are sufficient, the answer is very close to maximum (4.7 on a scale of 1-5). They also give a good score the areas of information, as well as images, text and buttons. But, do not completely agree (3.9) that the proposed pattern can help to improve the productivity of LOs. Based on this analysis we can conclude that new support tools for the construction of LOs are required. These news support tools should be easier to use, such as frameworks or templates, but keeping the design, and sub-areas of information.

Moreover, the teaching methodology (Educational Design) was not sufficiently well evaluated (4,05 in Table 3.), according to the authors, this is due to the absence of previous patterns which affects the phase II of the teaching methodology (Development).

Finally, the validation workshop concluded that to develop a low complexity LO, a lecturer required about three hours including pedagogical design.

V. ACKNOWLEDGEMENTS

The authors wish to thank the Research and Development Office of the Universidad Austral de Chile through project No. S-2007-15 entitled Learning Designs and Classroom Management, project DID S-2007-02 entitled Algebra Learning Units using ICT. Additionally wishes to acknowledge the valuable comments of all members of the Research Group on Learning Technologies (www.gita.cl).

REFERENCES

[1] Bradley, C. and Boyle, T. (2004). "Student evaluation of the use of learning objects in introductory programming". In L. Cantoni & C. McLoughlin (eds.), proceedings of ED-MEDIA 2004, World Conference on Educational Multimedia, Hypermedia & Telecommunications, June 21-26, Lugano, Switzerland, AACE, pp. 999-1006, ISBN 1-880094-53-3.

[2] Delgado J., Morales R., González S., and Chan M., (2007), "Desarrollo de Objetos de Aprendizaje basado en patrones". Sistema de Universidad Virtual, Universidad de Guadalajara. México. Retrieved

from http://ihm.ccadet.unam.mx/virtualeduca2007/pdf/228-JDV.pdf [last access:10.04.2010]

[3] Hernández, M., Alvarez. G., and Muñoz. J. “Patrones de Interacción para el Diseño de Interfaces WEB usables”. Retrieved from http://hciinterfazbuscador.iespana.es/PatronesInteraccionDiseno.pdf [last access:10.04.2010].

[4] Kujansuu, E. and Tapio, T. “Codewitz – An International Project for Better Programming Skills”. In L. Cantoni & C. McLoughlin (Eds.), Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2004 (pp. 2237-2239). Chesapeake, VA: AACE. Retrieved from http://www.editlib.org/p/12334 [last access:10.04.2010].

[5] Learning Technology Standards Committee of the IEEE. (2002): “Draft Standard for Learning Object Metadata. IEEE 1484”.12.1-2002. Online version. Retrieved from : http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf [last access:10.04.2010]

[6]. Miyadera, Y., Kurasawa, K., Nakamura, S., Yonezawa, N., and Yokoyama, S. “A Real-time Monitoring System for Programming Education using a Generator of Program Animation Systems” JOURNAL OF COMPUTERS, VOL. 2, NO. 3, MAY 2007. pp. 12-20

[7] Osorio, B., Muñoz, J., and Álvarez, J., (2007). “Metodología para el desarrollo de objetos de aprendizaje usando patrones”, 2da. Conferencia Latinoamericana de Objetos de Aprendizaje, Santiago de Chile (2007). Retrieved from : http://www.laclo.espol.edu.ec/laclo2007/index.php?option=com_content&task=view&id=26&Itemid=50&lang=en [last access:10.04.2010]

[8] Welie, M., (2000), “Patterns as Tools for User Interface Design”, Vrije Universiteit, Department of Computer Science, Amsterdam. The Netherlands, Retrieved from : http://www.welie.com/papers/TWG2000.pdf [last access:10.04.2010].

Pattern Catalog for Capability Diagnostics and Improvement of Service-oriented Enterprise Architectures

Alfred Zimmermann, Eckhard Ammann, Fritz Laux
Fakultät Informatik, business informatics research center
Reutlingen University
D-72762 Reutlingen, Germany
Email: {alfred.zimmermann | eckhard.ammann | fritz.laux}@reutlingen-university.de

***Abstract*—An original pattern catalog for capability diagnostics and optimization for change of service-oriented enterprise architectures is introduced. The current approaches for assessing maturity of software architectures were intuitively developed, having sparse meta model foundation and being rarely validated. This is a real problem because enterprise and software architects should know what is a successful path for introducing and changing service-oriented enterprise architectures.**

Our contribution is to extend existing Service Oriented Architecture (SOA) maturity models to accord with a sound meta model approach based on the well understood and standardized Capability Maturity Model Integration (CMMI), which was originally used to assess software processes and not architectures. Our specific architecture capability evaluation approach is the result of a meta model-based synthesis and conception and was grounded on the current Open Group Architecture Framework (TOGAF) standard for enterprise architectures.

Applying the maturity framework in consecutive assessment workshops with global vendors of service-oriented platforms provides the base for developing our pattern catalog for capability diagnostics and for improvement of service-oriented enterprise architectures.

***Index Terms*—Pattern catalog; SOA; SOAMMI; CMMI; TOGAF; service-oriented enterprise architecture; capability and maturity diagnostics; assessment; architecture maturity; meta model integration; maturity framework.**

I. Introduction and Related Work

Innovation oriented companies have introduced in recent years service-oriented architectures (SOA) to assist in closing the business and IT gap by delivering efficiently appropriate business functionality and integrating legacy systems with standard application platforms. Our approach of investigating the SOA ability of standard platforms in commercial use (see Buckow et al. [4]) assembles elements from convergent architecture methods and technologies like software related patterns as in Gamma et al. [11], Buschmann et al. [6], Fowler [10], and Buckl et al. [3], together with enterprise architecture management (EAM), SOA, and package based standard software applications. According to Alexander et al. [1] a pattern records the architecture decisions taken by many builders in many places over many years in order to resolve a particular problem.

The hypothesis of our research [18] is as follows:

1) The Capability Maturity Model Integration (CMMI) [7] is well known as suitable framework to assess software processes, nevertheless the meta model of CMMI can be extended to evaluate capabilities for change of enterprise and service-oriented architectures.
2) The idea of software patterns could be applied consistently for both capability diagnostics and for improvement of architecture areas starting from solid evaluation results of enterprise and service-oriented architectures.

The Open Group Architecture Framework (TOGAF) [17] as the current standard for enterprise architecture provides the basic blueprint and structure for our architecture domains:

- Architecture Strategy and Management,
- Business Architecture,
- Information Architecture,
- Application Architecture,
- Technology Architecture,
- Service & Operation Architecture, and
- Architecture Realization.

The Architecture Capability Maturity Model (ACMM) [2] framework, which is included in TOGAF, was originally developed by the US Department of Commerce. The main scope of ACMM is the evaluation of enterprise architectures in internal enterprise architecture assessments. The goal of ACMM assessments is to enhance enterprise architectures by identifying quantitatively weak areas and to follow an improvement path for the identified gaps of the assessed architecture. The ACMM framework consists of six maturity levels and nine specific architecture elements ranked for each maturity level - deviant from CMMI.

The SOA Maturity Model of Inaganti/Aravamudan [12] considers the following multidimensional aspects of a SOA: scope of SOA adoption, SOA maturity level to express architecture capabilities, SOA expansion stages, SOA return on investment, and SOA cost effectiveness and feasibility. The scope of SOA adoption in an enterprise is differentiated by following levels: intra department or ad hoc adoption, inter departmental adoption on business unit level, cross business unit adoption, and the enterprise level, including the SOA adoption within the entire supply chain. The SOA maturity levels are defined related, but different to CMMI using five ascending levels to add enhanced architectural capabilities: level 1 for initial services, level 2 for architected services, level 3 for business services, level 4 for measured business services, and level 5 for optimized business services. In a two-dimensional view - SOA scope and SOA maturity level - proper expansion stages for the systematic introduction of SOA in an enterprise are distinguished: fundamental SOA in a local department view, networked SOA with architected services on business unit level, and process enabled SOA on the enterprise level or in conjunction with suppliers. The SOA return on investment (ROI) increases gradually with increased maturity levels and matured SOA adoption. Shaded areas in the maturity model represent additionally no-go areas specifically the non-cost effective and the infeasible areas of SOA adoption.

The SOA Maturity Model from Sonic [16] distinguishes five maturity levels of a SOA, and associates them in analogy to a simplified metamodel of CMMI with key goals and key practices. Key goals and key practices are the reference points in the SOA maturity assessment.

The SOA Maturity Model of ORACLE [15] characterizes in a loose correlation with CMMI five different maturity levels - opportunistic, systematic, enterprise, measured, industrialized and associates them with strategic goals and tactical plans for implementing SOA. Additionally following capabilities of a SOA are referenced with each maturity level: Infrastructure, Architecture, Information & Analytics, Operations, Project Execution, Finance & Portfolios, People & Organization, and Governance.

Service-oriented architecture (SOA) is the computing paradigm that utilizes services as fundamental flexible and interoperable building blocks for both structuring the business and for developing applications. SOA promotes a business oriented architecture style, based on best of breed technology of context agnostic business services that are delivered by applications in a business focused granularity. A basic positioning into fundamental SOA concepts, technologies and case studies is offered by Erl [8] and for SOA-aspects on Enterprise Application Integration in the book of Krafzig et al. [13]. To provide agile composition of services within a worldwide environment and to enable flexible integration of published and discovered components, SOA uses a set of XML-based standards like WSDL, SOAP, UDDI and others. A main innovation introduced by SOA is that business processes are not only modeled but consistently used within a Model Driven Architecture (MDA) [14] approach to generate new and agile orchestrations or compositions of web services based on process diagrams. Early definitions of SOA were technology focused and the differences between SOA and web services were often blurred. SOA Technologies emerged due to the expansion of the Internet technology during the last years and produced an abundance set of specifications and standards developed by open standard organizations like W3C, OMG, OASIS, and The Open Group.

In the following Section II we provide our model synthesis for SOA Maturity Model Integration (SOAMMI) by assembling and shifting basic maturity model elements into our conceptual model for architecture maturity diagnostics. Based on the meta model of SOAMMI, Section III presents examples from our pattern catalog, which we have developed to assist in diagnostics and optimization of service-oriented enterprise architectures. Section IV states our conclusions and validation results from assessments and presents some ideas for future work.

II. SOA Architecture Maturity Framework (SOAMMI)

The aim of the SOAMMI was developed to provide a holistic framework to assess service-oriented enterprise architectures. The development process consisted of two interwoven phases. First, CMMI [7] was transformed from an assessment framework for software processes into a specific framework [18] to diagnose systematically the maturity of enterprise and software architectures.

Second, our maturity assessment approach was conducted by SOA applicators having experience in specific business domains and analyzing SOA vendor products for heterogeneous environments of legacy and standard applications. For the analysis we used assessment criteria, maturity domains, architecture capabilities, and level rankings from state of art SOA maturity models as described in [2] [12] [16] [15]. In addition specific architecture elements from [17] and [9] were selected to develop our architecture maturity model .

The SOAMMI architecture maturity framework introduces new architecture areas and organizes them within extended architecture domains, which are mainly based on TOGAF. Our intention was to leave most parts of the original CMMI meta model untouched and to extend the CMMI logic carefully.

The meta model of SOAMMI in Figure 1 has similarities with the CMMI meta model and defines additional specific elements, which are defined in the next sections for our architecture evaluation purpose. The extension uses maturity levels to measure the architecture maturity of vendor products in respect of requirements from customer oriented domain models:

- Maturity Level 1: Initial
  - Vendor service architecture is not performed or is incomplete or with no or initial coverage only
  - Architecture is unpredictable and poorly controlled
  - Initial service architecture methods and knowledge transfer about services and architectures
- Maturity Level 2: Managed
  - Vendor service architecture is managed, having medium completeness and coverage
  - Vendor supports learning about architectures and corrective actions are taken when necessary
  - Vendor service architecture is institutionalized within own products
- Maturity Level 3: Defined

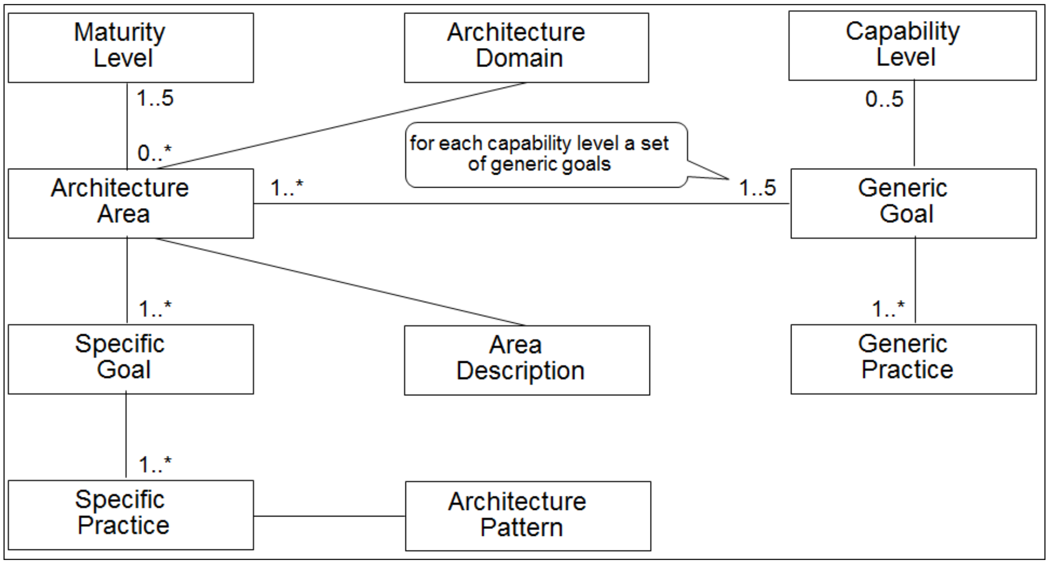


Fig. 1. SOAMMI Meta Model

  - Vendor service architecture is defined, having large, increasing completeness and coverage
  - Customer service architecture is agile tailored from standard vendor architecture
  - Vendor supports service strategy, architecture governance, methods and tools
- Maturity Level 4: Quantitatively Managed
  - Architecture artifacts and benefits are measured at vendor and customer side
  - Architecture is based on measured parameters from monitored business services
  - Causes of special variations are addressed
- Maturity Level 5: Optimizing
  - Defects are prevented at customer and vendor side
  - Innovations are added based on a vendor / client mutual roadmap
  - Change is expected, not feared and improvements are proactive.

Architecture domains were derived mainly from TOGAF [17], where they are used as specific architecture subtypes and corresponding phases of the ADM (Architecture Development Method). The top level structure of SOAMMI is organized by the following orthogonal architecture domains: Architecture Strategy and Management, Business Architecture, Information Architecture, Application Architecture, Technology Architecture, Service & Operation Architecture, and Architecture Realization.

Architecture areas are correspondent parts of process areas from CMMI. We have defined 22 specific architecture areas of SOAMMI in Figure 2 - fitting our architecture diagnostic scope, but different from CMMI - and structure them according to standard architecture maturity levels in line with the mentioned architecture domains. Each of the 22 delimited architecture areas are accurately described by a name and a short identification, and later on supplemented by a detailed description.

The following example of a standardized form shows in detail two specific architecture areas of the Business Architecture Domain, which were structured similarly to process areas of CMMI:

A. Architecture Area: BPS Business Products & Services

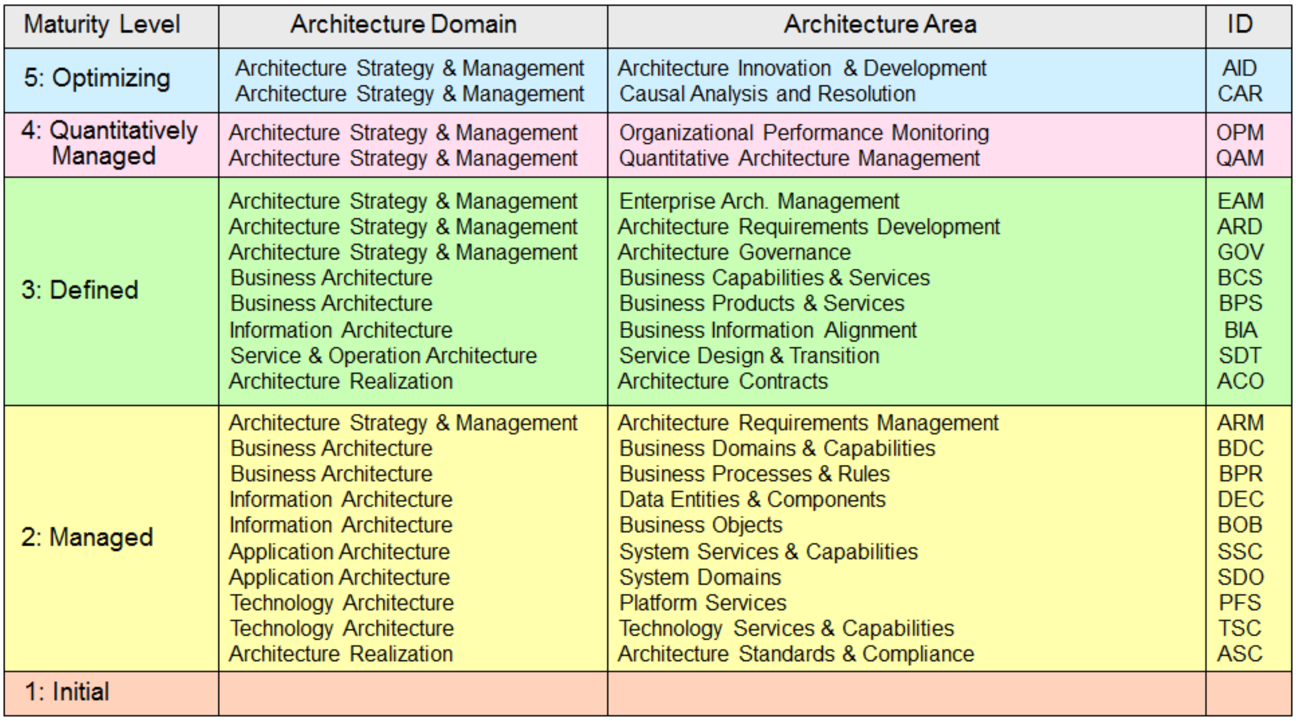
Purpose: Structure, design, model, and represent business products and associated business services, which are necessary to support modeled products.

Maturity Level: 3
Specific Goals (SG) and Specific Practices (SP):

- SG 1: Model Business Products as Origin of Business Processes
  - SP 1.1 Structure business products within product lines
  - SP 1.2 Design business products by defining product structures and product rules
  - SP 1.3 Model and represent business products
- SG 2: Model Business Services associated with Business Products
  - SP 2.1 Structure business services according product types
  - SP 2.2 Design business services by defining service structures and service levels
  - SP 2.3 Model and represent business services

B. Architecture Area: BPR Business Processes & Rules

Purpose: Structure, design, model, and represent business value chains and business processes to support modeled products and services.



Maturity Level	Architecture Domain	Architecture Area	ID
5: Optimizing	Architecture Strategy & Management	Architecture Innovation & Development	AID
	Architecture Strategy & Management	Causal Analysis and Resolution	CAR
4: Quantitatively Managed	Architecture Strategy & Management	Organizational Performance Monitoring	OPM
	Architecture Strategy & Management	Quantitative Architecture Management	QAM
3: Defined	Architecture Strategy & Management	Enterprise Arch. Management	EAM
	Architecture Strategy & Management	Architecture Requirements Development	ARD
	Architecture Strategy & Management	Architecture Governance	GOV
	Business Architecture	Business Capabilities & Services	BCS
	Business Architecture	Business Products & Services	BPS
	Information Architecture	Business Information Alignment	BIA
	Service & Operation Architecture	Service Design & Transition	SDT
	Architecture Realization	Architecture Contracts	ACO
2: Managed	Architecture Strategy & Management	Architecture Requirements Management	ARM
	Business Architecture	Business Domains & Capabilities	BDC
	Business Architecture	Business Processes & Rules	BPR
	Information Architecture	Data Entities & Components	DEC
	Information Architecture	Business Objects	BOB
	Application Architecture	System Services & Capabilities	SSC
	Application Architecture	System Domains	SDO
	Technology Architecture	Platform Services	PFS
	Technology Architecture	Technology Services & Capabilities	TSC
	Architecture Realization	Architecture Standards & Compliance	ASC
1: Initial			

Fig. 2. SOAMMI Architecture Areas and Maturity Levels

Maturity Level: 2
Specific Goals (SG) and Specific Practices (SP):

- SG 1: Model Business Value Chains as Root of Business Processes
  - SP 1.1 Identify business value for business operations
  - SP 1.2 Structure value chains
  - SP 1.3 Optimize business considering customer channels and supplier networks
- SG 2: Model and Optimize Business Processes
  - SP 2.1 Identify business activities for business processes: system activities, user interaction activities, manual activities
  - SP 2.2 Structure business processes for business roles and organizational units
  - SP 2.3 Define business workflows and business process rules
  - SP 2.4 Model and represent business processes
- SG 3: Model and Represent Business Control Information
  - SP 3.1 Identify and represent control information for product monitoring
  - SP 3.2 Identify and represent control information for process monitoring

The sketched Architecture Area BPS Business Products & Services was mapped as a premium architecture discipline to the higher Maturity Level 3 on top of the basic Architecture Area BPR Business Processes & Rules, which was allocated with the basic Architecture Maturity Level 2.

III. Enterprise Architecture Patterns

Our patterns for enterprise and service-oriented architectures consist of a set of methods which use best practices for diagnosing malfunctions and improving enterprise and information systems architectures. We have derived the methods from the structures of the metamodel of SOAMMI presented in Section II. Patterns, as described originally by Alexander et al. [1] are collections of best practices which are based on representing compactly core causalities for problem solving starting with a description of a recurring problem directing us to a standardized solution. Additionally to the core causalities for problem solving each pattern approach has added important but divergent extensions resulting in specific canonical forms for describing these patterns like in [11] [6] [10] [3].

Our pattern catalog for diagnostics and improvement of enterprise and service-oriented architectures organizes the collection of patterns according to the SOAMMI metamodel structures:

- Architecture Domains
- Architecture Areas
- Problem Descriptions associated with Specific Goals, and
- Solution Elements of the patterns connected to relate Specific Practices.

Linking solution elements to specific practices of the SOAMI Framework enables concrete solutions for diagnostics and improvement of service-oriented enterprise architectures. This diagnostic and improvement knowledge is no design knowledge, it is rather a procedural knowledge based on standards, best practices, and assessment experience for software and enterprise architectures. It is therefore both concrete and specific for setting the status of service-oriented enterprise architectures, and helping to establish an improvement path for change.

Patterns of our catalog show what to assess. Our patterns aim to represent diagnostic and improvement procedural knowledge to support cooperative assessment and improvement work of many people over many years in cyclic assessments of service-oriented enterprise architectures.

Associated with this pattern catalog we have set up an assessment process showing how to assess architecture capabilities. This process is based on a questionnaire for assessment workshops providing concrete questions and answer types, and helping to direct and standardize the related assessment process.

Additionally, we have included process methods for workshops, result evaluations, improvement path information for technology vendors and for application organizations, as well as change support and innovation monitoring instruments.

Based on the two Architecture Area examples from Section II- BPS Business Products & Services Architecture and BPR Business Process & Rules - we are deriving exemplarily the related subset of architecture patterns from the mentioned Specific Goals:

1) Model Business Products as Origin of Business Processes
2) Model Business Services associated with Business Products
3) Model Business Value Chains as Root of Business Processes
4) Model and Optimize Business Processes
5) Model and Represent Business Control Information.

We have chosen the reduced canonical form consisting of a succinct representation of the core causalities of our diagnostic and improvement patterns denominating consciously only the problem and the solution part as basic elements of our diagnostic and improvement patterns for service-oriented enterprise architectures. This basic canonical form of our currently used patterns is extendable by additional parts like contexts, examples, explanations, linked patterns, and others. We note that our diagnostic and improvement patterns are basically process patterns for enterprise architecture management and are therefore not fine granular classical design patterns. The following examples show a concrete extract from our set of 38 diagnostic and improvement patterns.

A. Example Pattern 1: Model Business Products as Origin of Business Processes

Problem: How can we structure, design, model, and represent business products as an origin for modelling business processes?

Solution:

- Structure business products within product lines
- Design business products by defining product structures and product rules
- Model and represent business products

B. Example Pattern 2: Model Business Services associated with Business Products

Problem: How can we structure, model, and represent business services needed to support business products?

Solution:

- Structure business services according product types
- Design business services by defining service structures and service levels
- Model and represent business services

C. Example Pattern 3: Model Business Value Chains as Root of Business Processes

Problem: How can we structure, optimize and represent value chains as roots for business process modelling?

Solution:

- Identify business value for business operations
- Structure value chains
- Optimize business considering customer channels and supplier networks

D. Example Pattern 4: Model and Optimize Business Processes

Problem: How can we structure, optimize and model business processes, related workflows, and business process rules?

Solution:

- Identify business activities for business processes: system activities, user interaction activities, manual activities
- Structure business processes for business roles and organizational units
- Define business workflows and business process rules
- Model and represent business processes

E. Example Pattern 5: Model and Represent Business Control Information

Problem: How can we model and represent business monitoring and control information?

Solution:

- Identify and represent control information for product monitoring
- Identify and represent control information for process monitoring.

The basic causality of our architecture pattern allows us to navigate in two directions: from the problem statement to the solution and backwards from the expected solution to the problem. From this navigation possibilities follow two important problem solving strategies for:

- Diagnostic: for verifying suggested solutions and defining the problem (from pattern solution to the pattern problem statement)

- Improvement: for identifying suitable solution elements for a given problem (from the pattern problem statement to the solution statements).

We have identified and distinguish a set of 38 Patterns in the context of 7 Architecture Domains and 22 Architecture Areas. The full list of patterns and its catalog structure follow the SOAMMI Framework [18]. From each mentioned 7 architecture domains we are sketching typical examples for enterprise patterns from our pattern catalog suited for architecture diagnostics and improvements:

1) Architecture Domain: Architecture Strategy and Management
   - Architecture Area: GOV Architecture Governance
     - Manage and control architectures of information systems
     - Support architecture governance
2) Architecture Domain: Business Architecture
   - Architecture Area: BPR Business Processes & Rules
     - Model business value chains as root of business processes
     - Model and optimize business processes
     - Model and represent business control information
3) Architecture Domain: Information Architecture
   - Architecture Area: BIA Business Information Alignment
     - Determine alignment of business and information architecture
4) Architecture Domain: Application Architecture
   - Architecture Area: SDO System Domains
     - Define and model a system domain map
5) Architecture Domain: Technology Architecture
   - Architecture Area: PFS Platform Services
     - Identify and model platform services from basic infrastructure
     - Determine fitness of vendor platform services
6) Architecture Domain: Service & Operation Architecture
   - Architecture Area: SDT Service Design & Transition
     - Identify and model services to support information systems and enable transition of services for support by service providers
     - Ensure service management offering for SOA
7) Architecture Domain: Architecture Realization
   - Architecture Area: ASC Architecture Standards & Compliance
     - Manage and control architecture standards and ensure compliance of architectures with standards
     - Support architecture standards, methods, and tools.

The practical benefits of our pattern catalog in the reduced canonical form is documented by the successful use as guideline for questionnaire design for two major capability assessments of service-oriented vendor technology architectures. Architecture assessments need to address the key challenges for companies during the built-up and management of service-oriented architectures in heterogeneous IT environments. Assessments of the SOA ability of standard software packages can be viewed additionally as a mean to engage with vendors on all relevant challenges of SOA in practical use.

Therefore, we did not design our assessment in form of a survey that could be filled out remotely, but rather focused on a discussion format where answers should include artifacts, cases, best practices, etc. As most questions have different relevance and meaning for different companies, our assessment is not intended to serve as a vendor ranking of any kind. These goals imply that a pragmatic simplification of SOAMMI is required, that needs to be enriched with specific user requirements from companies using SOA in heterogeneous environments.

Following these ideas, the basic structure of our questionnaire [5] was taken from SOAMMI architecture areas with one or more questions per specific goal respectively the problem statement in our diagnostic and improvement patterns. User requirements have been consolidated and mapped against specific goals. Wherever no user requirements could be mapped, specific practices or solution elements in our patterns have been used to generate questions on the level of specific goals. Through this procedure each specific goal could be related to at least one concrete question.

The assessment process takes about 3 months in total to complete for each software technology provider. The first step is a Pre-Workshop (2-3 hours) to make sure, that the vendor can identify the appropriate experts for the assessment workshop itself. Then the actual Assessment Workshop (4-6 hours) is held a few weeks later, so that the vendor has enough time to identify the experts that should participate and prepare answers. The SOA Innovation Lab (a consortium of SOA applicators, consulting companies, system integrators, and academic consultants) then prepares the summary of the findings and presents these to the vendor (1-2 hours). Finally, a series of follow up workshop for specific questions (3-4 hours each) is arranged with the vendor.

IV. Conclusion

A pattern catalog for diagnosing capabilities and improvement of organizational maturity of enterprise and service oriented architectures has been introduced. In this paper we have motivated the necessity to extend existing SOA maturity models to accord to a clear meta model approach due to the well understood and verified CMMI model. Based on the related work to CMMI, which is an assessment and improvement model for software processes, we have transformed and developed suitable models for the evaluation of SOA capability and maturity. Our specific architecture evaluation approach from the SOAMMI framework was founded on the current TOGAF standard for enterprise architectures. SOAMMI - The SOA Maturity Model Integration is the result of a meta model based conception and synthesis to provide a sound basis for practical evaluations of service oriented standard platforms in heterogeneous environments. Additionally a SOAMMI dashboard was developed to support practical assessment processes, which were aligned both with the SCAMPI process for CMMI and with empirical questionnaire and interview methods. The presented SOAMMI framework was validated in consecutive assessment workshops with two global vendors of service-oriented platforms and has provided transparent results for subsequent changes on service oriented product architectures and related processes. Our empirical validation and optimization of the presented maturity framework for its future usage is an ongoing process, which has to be synchronized with future cyclic evaluations of SOA platforms and their growing number of services. Extended validations of customers of service oriented technologies are planned for the next phase of our framework research and development. An idea towards a framework for individual enterprises is to generically extend the architecture areas to provide distinct views for architecture maturity diagnostics of vendor architectures and to support diagnostics for customers' and suppliers' abilities to handle service-oriented application architectures. Future work additionally has to consider conceptual work on both static and dynamic architecture complexity, and in connecting architecture diagnostic procedures with prognostic processes on architecture maturity with simulations of enterprise and software architectures.

References

[1] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S. (1977): A Pattern Language, Oxford University Press New York

[2] ACMM (2007): Architecture Capability Maturity Model - The Open Group. URL: http://www.opengroup.org/architecture/togaf9-doc/arch/chap51.html

[3] Buckl, S., Ernst, A.M., Lankes, J., Matthes, F. (2008): Enterprise Architecture Management Pattern Catalog. Technical Report TB 0801, sebis Technische Universität München

[4] Buckow, H., Groß, H.-J., Piller, G., Prott, K., Willkomm, J., Zimmermann, A. (2010): Method for Service-Oriented EAM with Standard Platforms in Heterogeneous IT Landscapes. Proceedings of the 2^{nd} European Workshop on Patterns for Enterprise Architecture Management (PEAM2010) Workshop at the Software Engineering 2010 Conference in Paderborn, February, 22 - 23, 2010

[5] Buckow, H., Groß, H.-J., Piller, G., Prott, K., Willkomm, J., Zimmermann, A. (2010): Analysing the SOA ability of Standard Software Packages with a dedicated Architecture Maturity Framework. (accepted paper) EMISA 2010 - Entwicklungsmethoden für Informationssysteme und deren Anwendung, Karlsruhe, FZI Forschungszentrum Informatik, October, 7 - 8, 2010

[6] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996): Pattern-oriented Software Architecture. Wiley

[7] CMMI (2006): CMMI for Development. Version 1.2, Carnegie Mellon University, Software Engineering Institute.
URL: http://www.sei.cmu.edu/reports/06tr008.pdf

[8] Erl, T. (2005): Service Oriented Architecture. Prentice Hall

[9] essential (2009): The Essential Architecture Project.
URL: http://www.enterprise-architecture.org/

[10] Fowler, M. ed. (2003): Patterns of Enterprise Application Architecture. Addison Wesley

[11] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994): Design Patterns. Addison Wesley

[12] Inaganti, S., Aravamudan, S. (2007): SOA Maturity Model. BP Trends, April 2007. URL: http://www.bptrends.com/publicationfiles/04-07-ART-The%20SOA%20MaturityModel-Inagantifinal.pdf

[13] Krafzig, D., Banke, K., Slama, D. (2005): Enterprise SOA. Prentice Hall

[14] MDA (2003): Model Driven Architecture (MDA) Guide OMG 1.0.1.
URL: http://www.omg.org/cgi-bin/doc?omg/03-06-01

[15] ORACLE SOA Maturity Cheat Sheet (2009)
URL: http://www.scribd.com/doc/2890015/oraclesoamaturitymodelcheatsheet

[16] Sonic (2005): SOA Maturity Model. Sonic Software Corporation, AmberPoint Inc., Systinet Corporation.
URL: http://soa.omg.org/Uploaded%20Docs/SOA/SOA_Maturity.pdf

[17] TOGAF (2009): TOGAF The Open Group Architecture Framework. Version 9.
URL: http://www.opengroup.org/architecture/togaf9-doc/arch/

[18] Zimmermann, A. (2009): SOAMMI - SOA Maturity Integration Model - Conceptual Framework. Research Study from Sabbatical at Daimler AG with SOA Innovation Lab, July 2009

Towards A Taxonomy of Dynamic Invariants in Software Behaviour

Teemu Kanstrén
VTT Technical Research Centre of Finland
Kaitoväylä 1, 90570 Oulu, Finland
teemu.kanstren@vtt.fi

Abstract— **The use of dynamic invariants to describe software behaviour has gained increasing popularity and various tools and techniques for mining and using these invariants have been published. Typically, these invariants are used to support various software engineering tasks, such as testing and debugging, which require one to understand and be able to reason about the system behaviour in terms of these invariants. However, the existing works are generally focused on a specific set of invariants for a specific purpose. In many cases it is also useful to view these in a wider context to enable a wider understanding of the invariants and to provide more extensive support across different domains. This paper presents work towards a general taxonomy describing the properties of dynamic invariants based on a review of existing work in their use, providing a basis for a wider adoption of different invariant features in different domains.**

Keywords-dynamic invariants; taxonomy; software behaviour

I. Introduction

Dynamic invariants are used to describe invariant properties of software behavior in terms of dynamic analysis. Dynamic analysis uses as basis information captured as observations from a (finite) set of program executions, such as test executions [1]. In line with these definitions, a dynamic invariant is defined here as a property that holds at a certain point or points in a program execution [2]. Recently the use of such dynamic invariants has become an increasingly popular technique in supporting different software engineering tasks (e.g., [3,4,5]).

Examples of dynamic invariants include data-flow constraints (e.g., `x` always greater than `0`) [2], control-flow constraints (e.g., `request` always followed by a `reply`) [6], or their combinations (e.g., `x` is always greater than `0` when `request` is followed by a `reply`) [7]. Invariants defined in terms of dynamic analysis can also be referred to as likely invariants as they are based on observations made from a set of program executions, which typically do not cover the entire program behavior state-space [2].

Dynamic invariants can be mined with automated tools or specified manually for further processing with automated tools. The idea of documenting and using invariants to reason about program behavior at run-time can be seen to be as old as programming itself [8,9]. Using invariants expressed in first-order logic to capture formal constrains on program behavior was introduced as early as 1960's [8] by the pioneering work of Floyd [10] and Hoare [11].

Dynamic invariants can be used in a variety of software engineering tasks and domains, such as helping in program comprehension [2,12], behavior enforcement [13], test generation and oracle automation [5], or debugging [14]. Thus, when explicitly defined, a set of invariants forms a basis for building automated support for many different purposes.

There exist a number of tools to support the use of dynamic invariants in different tasks [2,5,12]. Many of these tools use a specific set of invariants for a specific domain. When applying dynamic invariants in different domains, it is useful to also consider them in a wider context. Also, when a set of invariants needs to be provided, either as manually defined input for a tool to use as a basis for automated processing, or as output by an automated specification mining tool, being able to generally reason about them is needed for their effective use.

This paper describes a taxonomy for dynamic invariants. The taxonomy describes a set of common properties for invariants describing the dynamic properties of software behavior. As a basis, a set of invariants and their use have been reviewed from existing works. The study is structured to describe how the invariants are specified and used, what kind of invariant patterns over software behavior they capture, in which scope of behavior they apply, and what information about the system behavior is needed to be able to express and evaluate them.

The goal of this paper is to provide a starting point for a `road map' of the work accomplished so far on dynamic invariants, to provide help software engineers identify open research questions and new branches of discoveries, and to facilitate the use of dynamic invariants by a systematic definition of their different properties.

This paper is structured as follows. Section II describes the overall approach taken to create the taxonomy. Section III presents the taxonomy, its axes, and the individual categories. Finally, section IV provides discussion followed by concluding remarks.

II. Taxonomy Building Approach

Following guidelines from [15] for performing reviews, the works selected in this paper have been chosen where they describe or use some form of invariants over dynamic software behavior. This includes how these invariants are (manually) defined, and how automated specification mining approaches are used to produce them. For the sake of space and focus, this selection is focused on the originality of the work (in terms of adding to the taxonomy), its excellence (study process), and observed impacts (citation). Papers that take specific approaches to use and define invariants are also considered to provide a wider view. This approach is inspired and follows the taxonomy building approaches taken by Ducasse et al. [16] and Kagdi et al. [17].

The information presented in the taxonomy is based on information from publicly available works such as research papers, PhD theses and technical reports. Different publication databases were used as a basis for the search of related work. The focus of this paper is on properties related to invariants over dynamic behavior of software and thus the selection is focused on invariants over software runtime behavior. Work in the more static formal methods domain is reviewed when referenced from the works on dynamic behavior analysis but otherwise is not considered more deeply. An adapted version of Binder's ``fishbone" diagram [18] is used to describe the different aspects of the taxonomy.

In building the taxonomy, an initial version of the main axes and their classes was defined based on the seminal work of Ernst et al. [2] on defining dynamic invariants in terms of the program data-flow. This was then refined based on review of other works and how these contributed to evolving the taxonomy and its different properties. This resulted in a more advanced and more fully structured version of the taxonomy. This was presented to experts in the field (identified in the acknowledgements section). After this, additional refinement was done based on the comments received.

III. Taxonomy

This section describes the taxonomy that is the core contribution of this paper. The presentation starts with showing the main axes of the taxonomy, and proceeds to describe each axis in more detail in the following subsections.

A. *Axes of the taxonomy*

The main facets of the taxonomy of dynamic invariant properties of software behavior as discussed in this paper are presented in

Figure 1. This taxonomy is divided in six main facets, which are further divided to three process related ones and three facets describing information about the invariants themselves. The process-based facets describe various aspects of working with the invariants and include invariant specification, extraction, and usage. The invariant information facets are defining the invariants and include measurements, behavioral patterns, and scope. These six facets will be briefly presented here and discussed in more detail in the following subsections.

The generic flow of using invariants is presented in Figure 2. To make use of invariants, a set of invariants describing the aspects of interest in the software behaviour needs to be defined. This can be done either manually or with the use of an automated mining tool. In case a mining tool is used, a set of invariant templates describing a potential set of useful invariants is needed (e.g., [2]). An extensive basis for providing such templates this is provided by the invariant information properties of the taxonomy. The same applies for manual specification, as the properties of invariant information enable effectively reasoning about possible invariants. Analyzing software behaviour is typically based on large sets of observations (trace data), automated tools to help project the specified invariants over the captured observations is needed. This is again based on a similar tools and invariant templates as when using automated mining tools in the specification phase. For this reason, the specification and extraction phases are described in terms of shared properties in the following subsections of the taxonomy. However, from the process perspective, it should be noted that typically two phases of the process follow where a step of invariant specification is done and another step of extraction is done in order to form a basis for the final step of invariant usage, where these two are compared against each other.

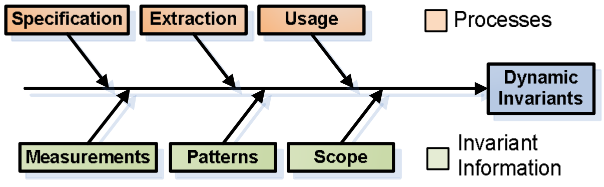


Figure 1. The taxonomy: Main facets.

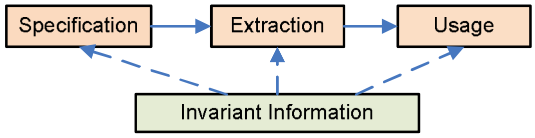


Figure 2. The taxonomy: Flow of elements.

Invariant information includes the measurements that describe the actual data that one needs to observe in order to build or evaluate an invariant, the behavioral patterns that the invariants describe over the measurements, and the scope of program behaviour when the invariant is expected to hold.

B. *Specification and Extraction*

The different aspects of invariant specification and extraction are illustrated in Figure 3. As mentioned before, these two phases share many properties and are thus described here in terms of common properties.

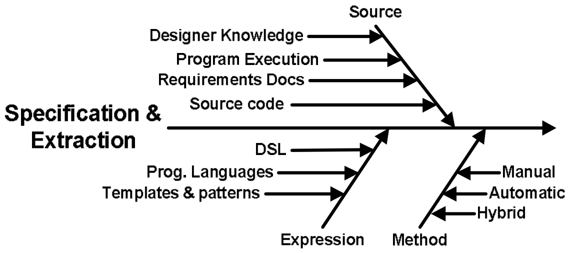


Figure 3. Invariant specification.

Different approaches to obtain the information for specifying the invariants include fully (**automatically**) reverse engineering these from observing program behaviour [2], describing them **manually** based on specifications or designer knowledge [19,5], or taking a **hybrid** approach where the reverse-engineered information is manually augmented with information from specifications [20]. When extracting a set of invariants based on dynamic analysis, the set of observed **program executions** is defined by what is available (e.g., test suite) and what is the goal of the analysis (e.g., analysis of a subset of the entire test suite) [1].

The type of specification is closely related to the source of the information used for the specification. Natural language documents such as **requirements specifications** can be manually analysed to find a set of relevant invariants [20]. Additional **designer knowledge** gained when working on the system provides insights into its invariant behaviour, allowing them to design additional invariants to describe the implementation [19]. Source code and program execution are more suited for automated analysis. As **source code** is mainly useful in terms of static analysis, it can be used mainly as an additional input for dynamic analysis such as providing interface definitions [21,20]. From these sources, one needs to capture a set of suitable invariants describing the relevant properties of dynamic behaviour in the software. Experiments have shown that combining both types of sources gives the best results, where both provide useful invariants not identified by the other approach [22].

When the invariants have been defined, they need to be expressed in order to allow the user to process them effectively. Domain specific languages (**DSL**) can be used to describe the invariants specifically for a chosen domain, such as in form of test oracles for web-applications [5]. **Templates and patterns** can be created to express a chosen set of invariants generally over different programs and used for automated invariant mining [2,14]. **Programming languages** provide powerful constructs that can be used to describe invariants as they allow for full specification of all properties expressable in a full programming language [19].

C. *Invariant Usage*

The usage domain of an invariant refers to the context and goal to which it is applied. This describes both the usage domains describing what the invariants are used for and usage types describing if they are applied in an operational system or separately from it. The invariant usage aspects of the taxonomy are illustrated in Figure 4.

In **offline use**, the invariants are used separately from the execution of the analysed program. In **online use** the usage of the invariants is linked to the executing program. Behaviour enforcing techniques guide the online operation of the observed system. **Static analysis** is focused on automated analysis of given static artifacts and thus mainly operate offline. Besides these two, the other domains make equal use of both online and offline approaches. Since this paper is focusing on dynamic analysis, static analysis is not considered further here other than to note that dynamic invariants can also be used as input for it [23].

Behaviour specification is the basic relevant concept for any application of invariants described in this paper, as the invariants need to be specified before they can be used. In itself, this does not constitute as a usage domain but rather as a basis for the other approaches. From the specification perspective, these invariants can also be used as the basis for application of many formal methods such as model checking and other forms of static analysis [6,24]. For example, a data-flow invariant can specify that a return value should always be greater than zero [24], or that the value returned by get() should always match the last given parameter of set() [25]. Similarly, control-flow related invariants can be used to define constraints such as always closing opened database connections [26].

A specific area in the domain of behaviour specification is the automated mining of specifications based on dynamic invariants. Various tools that work with dynamic invariants are in fact aimed at automatically mining specifications in terms of invariants for the user to process [2,4,7,12,14,26,27,28,29,30]. In this sense, dynamic invariants are also used to assist in the process of specifying the software itself. However, the taxonomy described in this paper takes no stand on how the invariants are obtained. The taxonomy is intended to support the process of using and creating the invariants, whether through automated mining techniques or by manual specification. In both case, a systematic description provided by the taxonomy should help in creating and using them.

Behaviour analysis supports either automated or manual analysis of software runtime behaviour. A set of specified invariants are given and used in each case to analyse how the system behaves. This information is presented to the human user for analysis. Failure cause location can be supported by analyzing how the invariants change over time and reporting any significant changes before a failure is observed [4,14] and by comparing the invariants observed over both failing and non-failing program executions [4]. Software evolution tasks can be supported by presenting any changes over given invariants when changes are made to a program to make the impacts of changes more explicit [2,31], such as changed interaction sequences and input-output transformation [31]. Another example in this domain is suggesting refactoring based on invariants holding over values (e.g., parameter always constant) that can be used to simplify the program [32].

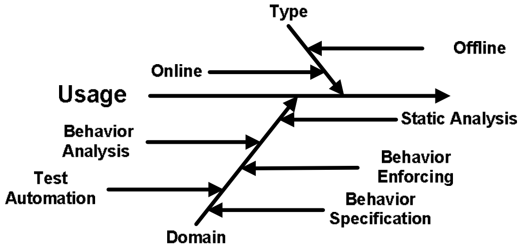


Figure 4. Invariant usage.

Further, in security assurance, observing a set of core invariants over specific variables, such as kernel data structures or session state variables can be used to identify potential security attacks when the expected invariants are violated [28,33]. Additionally, the invariants can support tasks such as program comprehension by providing a documentation that describes the software behaviour in terms of its important (invariant) behaviour [2].

Behaviour enforcing mechanisms analyse the behaviour of the observed software based on a given set of invariants similar to the domain of behaviour analysis. However, they additionally take automated action to modify the behaviour based on differences observed with regards to the given (expected) invariants. For example, automatic adaptation mechanisms can use invariants to choose a new state for the software based on which specified invariants hold at different points in time [13]. Invariants can also be used to ensure that failure states specified in terms of invariants are avoided by modifying runtime behaviour that is observed to be outside the given set of invariants (the expected behaviour) to fit inside the expected invariants [34,35].

Software **test automation** is basically a comparison of the expected behaviour of the software to its actual behaviour. This comparison is done by a test oracle that needs a representation of the expected behaviour of the software in terms of input-output transitions. As this needs to be described in terms of invariant behaviour, dynamic invariants can be used to encode this information as a basis for the test evaluation, where test results are expected to conform to these invariants [5,14,36,37].

When the invariants describe meaningful (important) properties of the software behaviour, they also make good candidates for evaluating which parts of the software behaviour should be covered in testing. Invariants can then be used to assess test coverage in terms of invariants covered by the test suite [9,38]. This can further be improved by automatically generating test inputs that aim to increase the set of covered invariants [37,39].

Component upgrade checking is a special type of test automation. In this context it is important to verify that an update of a component works with the rest of the system. Invariants can be used to describe how the component behaves with the other components, and to assess the relations of the invariants of the different components against each other. These invariants can describe, for example, the inputs and outputs of the different components in terms of control- and data-flow [25,31]. The comparison is then an evaluation of these invariants over the different versions.

D. Measurements

In order to apply dynamic invariants one needs to collect the required information to either assess that they hold or to infer (mine) them, depending on the intended usage domain. In any case, one needs to be able to define and collect the required information from the observed system. The process of extracting this information is referred to here as information extraction, similar to [40]. This aspect of the taxonomy is shown in Figure 5.

The **information type** of the measurements can be classified to two different types of static and contextual information [27]. **Static information** in a dynamic setting is information that is always the same for a given point of observation. For example, during a specific point of execution, a message passed can always be the same type of a message (e.g., method call named publishData()) and is thus static over different executions of this point. **Contextual information** described dynamic information that changes over the program executions over a single point depending on the context (e.g., test case) of the observed information. For example, the time of observation, parameter values, and the thread of execution for a given message all can change over different executions of the same program point [27,41]. The set of observations can also be grouped ("sliced") according to their contextual information, such as process (thread) id to produce a set of invariants over the scope represented by that slice [12,27,41]. In this case, the scope identifier becomes the basic measure (e.g., thread id [29] or constant parameter value [27]).

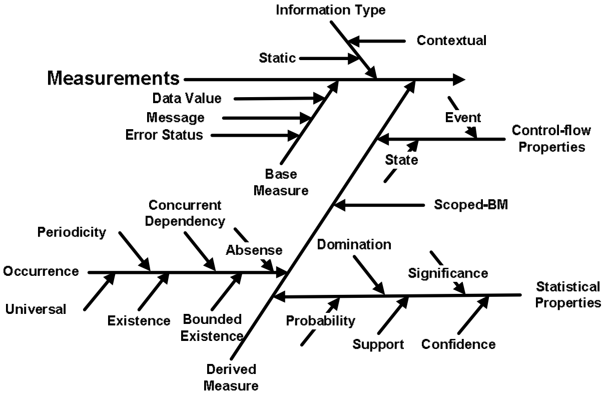


Figure 5. Measurements for invariants.

The term **base measure** is used here to refer to a type of measurement information that describes some basic value of program behaviour as it is observed. For dataflow variables this includes the **data values** with basic data types such as Boolean values, integers, and text strings (character sequences) stored in different variable and parameter values [2]. In the scope of object oriented programs the runtime type of an object can also be used as a base measure [14].

From the control-flow perspective the base measures are the **messages** passed between different elements of the control-flow. For example, method invocations between components (such as classes or services) [7,31] or invocations on graphical user interface (GUI) operators [5,36].

A specific case of control-flow is error handling flows identified by some **error status**. Error scenarios can be classified to generic errors and application specific errors [5]. Generic errors can be related to properties shared by different applications such as database access errors and user-interface (e.g., HTML or DOM tree for a web-application[5]) error codes. When represented in a uniform way (e.g., by programming language exception mechanisms[37]), these can be generally observed and described in the system behaviour (e.g., by an automated tool supporting a given domain). For example, all Java exceptions can be taken to describe a message that denotes erroneous behaviour being observed [37]. Application specific errors need to be described separately for each application in terms of application specific invariants. For example, one may expect a given error response to a message outside a given set of input [20].

A **derived measure** is something that is not directly observed in the system behaviour, but the value of which is rather derived from one or more base measures. To produce derived measures for data-flow, the base measures for a system can be grouped based on invariant scopes [2]. For example, the values of variable x before and after a program point can be considered separately as variables x1 and x2, to describe a pattern saying x1>x2. These produce **scoped**

base measures. The different scopes are discussed in subsection F.

A specific case of this is software **control-flow properties** that can be described in terms events and states. From the control-flow point of view, an **event** can be described as an identifiable, instantaneous action in the observed software behaviour, such as passing a message or committing a transaction [29]. Similarly, a **state** can be described as values of properties that hold over time, such as over interactions between components. This information can be, for example, held in interaction parameters or inside components internal state variables [33]. A related property is branching, which defines how several different paths of events and states can be taken in the software behaviour. This can be described in terms of invariants when observing which paths are taken and which ones are not [4,42].

Two properties related to both data- and control-flow measures are those describing their occurrence and statistical properties. Derived measures related to **occurrence** describe how data- or control-flow measures are expected to occur in a given scope. Of these, **absence** defines an expectation that the measure does not exist in the defined scope [6]. **Existence** denotes that the measure exists in a scope, and **bounded existence** that the measure exists N time in a scope, where N denotes either exact, minimum or maximum number [6]. **Universal** defines an expectation that the measure applies to the whole scope [6]. **Concurrent dependence** is related to observed fork and join points in execution [43]. A fork expects one measure to be followed by several measures of a given type and a join expects several measures of a given type to be followed by a single specific measure [43]. **Periodicity** describes a measure repeating over a given cycle (scope) [43].

Statistical properties describe additional information for other base- or derived-measures. Support and confidence are two values commonly used together. **Support** defines the number of times a measure is observed in behaviour [26,44]. **Confidence** can be used with the same definition [2] but also as a definition of how often another measure is observed in relation to support, meaning how often a precondition is followed by a post-condition [27,44].

Probability defines the threshold for a measure to be observed in a given scope. This can be used in different ways. A measure with low probability (support percentage) can be excluded from analysis to address anomalies [2,12,29]. Different approaches are used for this depending on the target invariants, from low level (1% or less) [2] to 20% [12]. The probability can also refer to probabilities of a measurement value inside a range of allowed values [13]. Deviations from the expected values are typically given a probability, which can then define the significance of the deviation [13,14,33]. This threshold can be used for different purposes such as identifying probable failure causes [14], security attacks [33], and to decide new states for automated adaptation [13].

Significance defines the importance of an invariant violation or of the measured variable. Different approaches to significance can be taken where the latter observed violations are given higher priority as they are seen to be closer to a failure [14], or earlier violations as they are expected to have more impact on latter behaviour [34]. When a variable is observed as having no correlation with other variables it can be considered irrelevant [32]. A generic derived measure used for these is the number of measurements. **Dominance** is a measure used to remove overlapping patterns where one includes the other as a sub-pattern [30].

E. Behavioral Patterns

A dynamic invariant in software behaviour basically describes a pattern over the observed behaviour. This aspect of the taxonomy is shown in Figure 6. Control-flow related patterns describe ordering of events or states in the observed system [6]. Data-flow related patterns describe the data-flow of the observed software, such as what values a given variable takes during the software execution [2].

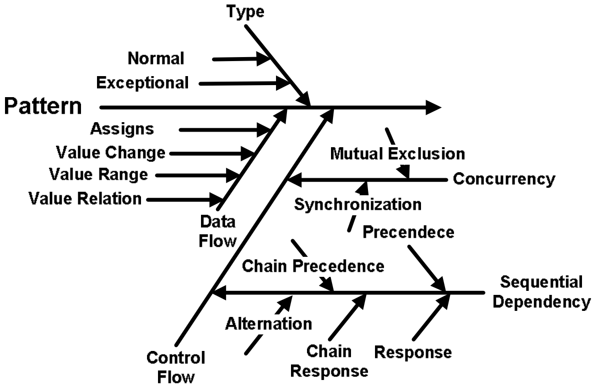


Figure 6. Behavioral patterns.

Together these can be combined to represent the complete behaviour of the software in terms of the control-flow combined with the data-flow. A basic way to describe these combinations is in terms of conditional dependence; a control-flow event can only be followed by one of many (branches) depending on a given condition [43]. A natural way to express these conditions is then in terms of invariants related to the data-flow in the context of that control-flow. For ex-ample, event `P1` can be followed by event `P2` when `x<0` and by `P3` when `x>=0`. Together these are referred to here as behavioral invariants, where the constraints for a given control-flow pattern are defined in terms of its data-flow invariants. For example, a stack allowing three pop operations after having three push operations performed on it [30]. These can be further combined to form a more complete model such as an extended finite state machine, where states represent the control-flow and the transitions between states are defined in terms of data-flow [20,34].

Each pattern can further be related to describing different types of behaviour, which can be generally classified as **exceptional** (error) or **normal** (correct) behaviour of the observed system [5]. For example, a transaction may complete or fail due to its parameters and environment state. As described in subsection D, different base measures related to errors can be used to identify them.

Control flow patterns basically describe the sequential dependencies between a programs events and states [43]. In the following discussion the term ``event" is used to refer to both events and states.

Alteration describes two or more events being grouped together and always appearing as an alternating sequence such as ABABAB [30,41]. Specialized cases can also be defined such as events in the alternating sequence repeating multiple times, AB*C, where B is repeated 1-N times between A and C [30], or a cutoff in the end of the sequence (ABABA) [12].

Precedence describes a specific event P always occurring before another specific event Q [6]. This can also be referred to as a precondition [24] and is a specific case of **chain precedence**, which defines that a sequence of events (Q1,Q2,Q3,...) is always preceded by another sequence of events (P1,P2,P3,...) [6].

The opposite of precedence is **response,** which defines that event P is always followed by event Q. This is again a specific case of **chain response**, which defines that a sequence of events (P1,P2,P3,...) is always followed by another sequence of events (Q1,Q2,Q3,...) [6].

Specific cases of alteration are the patterns related to **concurrency**. These can be classified to two main patterns of mutual exclusion and synchronization [29]. **Mutual exclusion** occurs when no two measures are observed at the same time. **Synchronization** has two specific cases, where two measures are always observed together (overlapping) or where one starts as another ends.

Data flow patterns describe properties and relations over variable values during program execution. The **assigns** pattern defines that in a defined scope, values of specific variables are assigned to (modified) [24]. This can also be described in terms of values that are not modified [2]. **Value change** is an evolutionary pattern that describes how a value changes over time in a given context. This pattern defines a reference value for the expected value distribution of the observed variable in the given scope. For example, the expectation can be that change in value is always small (within a given threshold such as change<5) [14]. A specific case can be a variable that is never set [2].

A **value range** describes a variable always having a range of values in a given scope [2,14]. Examples include value always being constant, one of a set of possible values (e.g., one of 1,2,4) and a value between given boundaries (e.g., 1<x<4) [2]. Common constants such as zero or one can also be considered a specific case in itself [2,14]. Additionally, the maximum and minimum can be considered [14]. Optimizing for performance a subset can also be selected such as looking for positive (x>0) or negative values (x<0) [14]. Another example is that the contents of a character string are expected to be a human readable character with a given probability distribution in how often each character is expected to be observed [33].

These can be seen as a special subset of the **value relation** pattern. A value relation describes how one variable is related to another [2,25]. These can be basic mathematical operations (e.g., x<y or x=y+1), or more complex mathematical functions [2]. Relations can also be described in terms of the relation of one variable to several others [25]. One example of this is the relation of program output to all of its (several) inputs [25]. In the case of larger sets of values (e.g., arrays), the same relations can be described internally between the elements of the set [2]. Additionally, a set of specific relations can be considered such as one set reversing another or matching a subset of a bigger set [2]. Additionally, a single value (e.g., a given variable or a constant) can be described to always be included in a given set [2].

F. Invariant Scope

The scope of an invariant defines where this invariant is expected to hold. The scope element of the taxonomy is shown in Figure 7. In the following descriptions, the term event is used to refer to both control-flow events and states and data-flow measures.

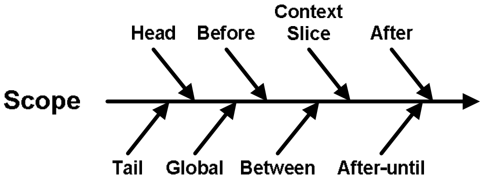


Figure 7. Invariant scope.

An invariant may define that it should hold **after** a given event [6]. Additionally, another event may be defined as the end condition in which case the invariant should hold after the observed start event until the observed end event (**after-until**) [6]. This is similar to the scope **between**, which defines two events in between which the invariant pattern should hold [6]. However, the difference is that this only holds once both the start and end events have been observed, and after-until holds from the first observation of the start event.

As opposed to the after scope, an invariant pattern can also be defined to only hold **before** a given event is observed [6]. A **global** invariant pattern should hold for all observed behaviour during the program execution [6]. Considering only the first N (**head**) or the last N (**tail**) observations of a set can also define a meaningful scope [2]. For example, the relations between the last 2 observations can define how a value in a set increments [2].

The scope can also be defined in combination with a specific slice of the program behaviour, such as a thread [41,27] or a specific web application session [33]. In this case the scope becomes a combination of the context **slice** and one of the other scope definitions discussed above.

IV. Discussion

The taxonomy and its classes presented above are based on the existing work in the literature. In this sense it limits itself to discuss properties only relevant to those in the chosen works. Additionally, it is possible to use and explore other possible relations. For example, many of the described control-flow patterns also apply to data flow patterns. For example, a value may be defined to precede another value (relating to the precedence control-flow pattern). Similarly, the set of data-flow patterns can be considered to apply in the context of control-flow. For example, the range of possible control-flow options following one control-flow event can be in a given range of possible defined control-flow events or states (related to value-range data-flow pattern).

The discussion in this paper is from the generic viewpoint of using dynamic invariants. One important aspect to consider is how representative the available invariants are in describing the relevant properties of software behaviour. When defined manually by an expert, the invariants can be expected to describe relevant and important properties. However, even in these cases important invariants can be missing and in many cases no invariants are defined at all. In these cases, automated inference techniques can be used to assist in finding invariants. Both of these cases have been shown to be valid as also discussed in section III.B. Improving the means to help manually define invariants and to automatically mine for relevant ones thus is an interesting research question. Potential approaches to investigate include using a set of chosen invariants known to be interesting in the given domain, using combined information from static analysis, relying on statistical values to report the more interesting ones, and providing more advanced support for combining both the manual and automated approaches as also discussed in section III.B.

Discussion on the statistical properties of different patterns and measures highlight differences in the applied approaches. For example, in many cases the invariant patterns that have only low support level (i.e., there are few cases) are only reported. In the extraction phase, this can be useful in removing patterns observed merely due to chance that may be incorrect in themselves due to interleaving of concurrent behavior, or completely irrelevant in the general context [2]. On the other hand, sometimes all observed behavior is important regardless of their probability. This can be, for example, behavior that is only rarely observed in the observed executions but is still equally important for the overall system behaviour (e.g., error handling or corner cases) [20].

Use of invariants in different domains as discussed here is not limited to those aspects discussed. In fact, many systems use invariants for various purposes but these are not always called invariants. For example, in test automation the test oracle practically always needs to be described in terms of an invariant, where the input is expected to produce a given output (the relation of input to output should be invariant). In this sense, defining invariants as discussed here can be beneficial in a wider context of how people think about the behaviour of programs. However, presenting a meaningful language to describe the invariants and use them in different contexts is required for adopting them as a concept more widely as many are not used to thinking in these terms.

Understanding and using invariants generally requires specific considerations for specific usage purposes. For example, one may refactor code based on suggestion from invariant analysis [32] but this also needs to consider the part where the human user needs to read the code and understand it. If the refactoring reduces this understanding by hiding information, this refactoring may be more harmful for the overall software maintenance. Similar needs for understanding the invariants in general need to be considered.

V. Conclusions and Future Work

Today, dynamic invariants are used for many points in software design and analysis. The invariants for different system are as different as their behaviour, but this paper has collected a set of common properties from existing works and presented a taxonomy describing these common properties. This should help give a more common understanding of dynamic invariants in software behaviour and help in using them in different domains.

The presented taxonomy is based on six main facets, three related to processes of using the invariants and three related to the information describing the invariants themselves. The main focus was on describing the properties of the invariants themselves, and thus on the parts describing the invariant information in the context of the process.

The main contribution of this paper is presenting the underpinning of a classification overview for understanding the space of dynamic invariants. This provides a basis for more thorough reasoning about invariants, building tool support and identifying future research questions. Some specific questions identified include possibilities of providing more focused domain specific invariants on top of the taxonomy and providing more extensive tool support for using the invariants according to the taxonomy presented, as existing tools only consider parts of it.

Topics for future work include further exploring the different aspects of dynamic invariants and their relations to each other, such as scopes, patterns and measurements. Similarly, a deeper investigation of their relation to other formalizations of software behavior, such as those used in the formal methods community is seen as interesting. Applications of the taxonomy along with the further investigations are also needed for practical validation and evolution.

Acknowledgment

The author wishes to thank Ali Mesbah and Arie van Deursen for their helpful comments on this paper and on the taxonomy.

References

[1] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," *IEEE Transactions on Software Eng.*, 2009.

[2] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically Discovering Likely Program Invariants to Support Program Evolution," *IEEE Transactions on Software Eng.*, vol. 27, no. 2, pp. 99-123, Feb. 2001.

[3] M. Boshernitsan, R. Doong, and A. Savoia, "From Daikon to Agitator: Lessons and Challenges in Building a Commercial Tool for Developer Testing," in *Int'l. Symposium on Software Testing and Analysis*, 2006, pp. 169-179.

[4] T. M. Chilimbi, B. Liblit, K. Mehra, A. V. Nori, and K. Vaswani, "HOLMES: Effective Statistical Debugging via Efficient Path Profiling," in *31st International Conference on Software Engineering*, 2009, pp. 34-44.

[5] A. Mesbah and A. van Deursen, "Invariant-Based Testing of Ajax User Interfaces," in *31st International Conference on Software Engineering*, 2009.

[6] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specifications for Finite-State Verification," in *21st International*

Conference on Software Engineering, 1999, pp. 411-420.

[7] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic Generation of Software Behavioral Models," in *30th International Conference on Software Engineering*, 2008, pp. 501-510.

[8] L. A. Clarke and D. S. Rosenblum, "A Historical Perspective on Runtime Assertion Checking in Software Development," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 3, pp. 25-37, 2006.

[9] D. Schuler, V. Dallmeier, and A. Zeller, "Efficient Mutation Testing by Checking Invariant Violations," in *18th Int'l. Symposium on Software Testing and Analysis*, 2009, pp. 69-80.

[10] R. Floyd, "Assessing Meaning to Programs," in *Symposium on Applied Mathematics, American Mathematical Society*, 1967, pp. 19-32.

[11] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576-580, 1969.

[12] D. Lo and S. Khoo, "SMArTIC: Towards Building an Accurate, Robust and Scalable Specification Miner," in *14th Int'l. Symposium on Foundations of Software Engineering*, 2006.

[13] L. Lin and M. D. Ernst, "Improving the Adaptability of Multi-Mode Systems via Program Steering," in *Int'l. Symposium on Software Testing and Analysis*, 2004.

[14] S. Hangal and M. Lam, "Tracking Down Software Bugs Using Automatic Anomaly Detection," in *24th International Conference on Software Engineering*, 2002.

[15] B. Kitchenham, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," Keele University, Keele, Staffs, EBSE Technical Report 2007.

[16] S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," *IEEE Transactions on Software Eng.*, vol. 35, no. 4, pp. 573-591, 2009.

[17] H. Kagdi, M. L. Collard, and J. I. Maletic, "A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution," *Journal of Software Maintenance and Evolution*, vol. 19, no. 2, pp. 77-131, 2007.

[18] R. V. Binder, "Design for Testability in Object-Oriented Systems," *Communications of the ACM*, vol. 37, no. 9, pp. 87-101, September 1994.

[19] Bertrand Meyer, "Applying Design by Contract," *Computer*, vol. 25, no. 10, pp. 40-51, 1992.

[20] T. Kanstrén, *A Framework for Observation-Based Modelling in Model-Based Testing*. Oulu, Finland: VTT, 2010.

[21] Johannes Henkel and Amer Diwan, "Discovering Algebraic Specifications from Java Classes," in *17th European Conference on Object-Oriented Programming*, 2003.

[22] Nadia Polikarpova, Ilinca Ciupa, and Bertrand Meyer, "A Comparative Study of Programmer-Written and Automatically Written Contracts," in *18th Int'l. Symposium on Software Testing and Analysis*, 2009.

[23] Jeremy W. Nimmer and Michael D. Ernst, "Invariant Inference for Static Checking: An Empirical Evaluation," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 6, 2002.

[24] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. Leino, and E. Poll, "An Overview of JML Tools and Applications," *International Journal in Software Tools for Technology Transfer*, vol. 7, pp. 212-232, 2004.

[25] S. McCamant and M. Ernst, "Early Identification of Incompatibilities in Multi-Component Upgrades," in *18th European Conference on Object-Oriented Programming*, 2004, pp. 440-464.

[26] S. Thummalapenta, T. Xie, N. Tillmann, J. de Halleux, and W. Schulte, "MSeqGen: Object-Oriented Unit-Test Generation via Mining Source Code," in *7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2009.

[27] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: Mining Temporal API Rules from Imperfect Traces," in *28th International Conference on Software Engineering*, 2006.

[28] A. Baliga, V. Ganapathy, and L. Iftode, "Automatic Inference and Enforcement of Kernel Data Structure Invariants," in *Annual Computer Security Applications Conference*, 2008, pp. 77-86.

[29] J. E. Cook and Z. Du, "Discovering Thread Interactions in a Concurrent System," *Journal of Systems and Software*, vol. 77, no. 3, pp. 285-297, Sept. 2005.

[30] M. Gabel and Z. Su, "Javert: Fully Automatic Mining of Temporal Properties from Dynamic Traces," in *16th Int'l. Symposium on Foundations of Software Engineering*, 2008.

[31] L. Mariani, S. Papagiannakis, and M. Pezzé, "Compatibility and Regression Testing of COTS-Component-Based Software," in *29th International Conference on Software Engineering*, 2007.

[32] Y. Kataoka, M. Ernst, W. Griswold, and D. Notkin, "Automated Support for Program Refactoring Using Invariants," in *International Conference on Software Maintenance*, 2001, pp. 736-743.

[33] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications," in *10th Int'l. Symposium on Recent Advances in Intrusion Detection*, 2007.

[34] D. Lorenzoli, L. Mariani, and M. Pezze, "Towards Self-Protecting Enterprise Applications," in *Int'l. Symposium on Software Reliability*, 2007, pp. 39-48.

[35] J. H. Perkins, G. Sullivan, W. Wong, Y. Zibin, M. D. Ernst, M. Rinard, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, f. Sherwood, and S. Sidiroglou, "Automatically Patching Errors in Deployed Software," in *ACM SIGOPS 22nd Symposium on Operating System Principles*, 2009.

[36] A. M. Memon, "An Event-Flow Model of GUI-based Applications for Testing," *Journal of Software Testing, Verification and Reliability*, vol. 17, pp. 137-157, 2007.

[37] C. Pacheso and M. D. Ernst, "Eclat: Automatic Generation and Classification of Test Inputs," in *European Conf. on Object-Oriented Programming*, 2005, pp. 504-527.

[38] M. Harder, J. Mellen, and M. D. Ernst, "Improving Test Suites via Operational Abstraction," in *International Conference on Sofware Engineering*, 2003, pp. 60-71.

[39] T. Xie and D. Notkin, "Tool-Assisted Unit-Test Generation and Selection Based on Operational Abstractions," *Journal of Automated Software Engineering*, vol. 13, no. 3, pp. 345-371, July 2006.

[40] C. Ackermann, M. Lindvall, and R. Cleaveland, "Recovering Views of Inter-System Interaction Behaviors," in *16th Working Conference on Reverse Engineering*, 2009, pp. 53-61.

[41] J. E. Cook and A. L. Wolf, "Discovering Models of Software Processes from Event-Based Data," *ACM Transactions on Software Engineering and Methodology*, vol. 7, pp. 215-249, 1998.

[42] N. Kuzmina, J. Paul, R. Gamboa, and J. Caldwell, "Extending Dynamic Constraint Detection with Disjunctive Constraints," in *Int'l. Workshop on Dynamic Analysis*, 2008.

[43] J. E. Cook and A. L. Wolf, "Event-Based Detection of Concurrency," in *6th Int'l. Symposium on Foundations of Software Engineering*, 1998, pp. 35-45.

[44] D. Lo, L. Mariani, and M. Pezze, "Automatic Steering of Behavioral Model Inference," , 2009, p. 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.

Towards a Common Pattern Language for Ubicomp Application Design - A Classification Scheme for Ubiquitous Computing Environments -

René Reiners
Fraunhofer FIT
Schloss Birlinghoven
53754 Sankt Augustin
rene.reiners@fit.fraunhofer.de

***Abstract*—The idea of *Ubiquitous Computing* was first formalized and described by Mark Weiser in the early 90's. Since then, it has been followed by many research groups and extended in many ways. There are many ideas, concepts, prototypes and products implementing ubiquitous computing scenarios. However, the manifold of approaches also brings along a large variety of denominations for eventually similar concepts. Our work seeks for the creation of a dynamic pattern language gathering design knowledge for ubiquitous computing applications and the underlying concepts. The intention is to support researchers and application designers in the domain to avoid the repetition of design errors and provide design knowledge about successful approaches. In order to get closer to that aim, our first step is to present a classification scheme applicable to existing and future approaches that is needed in order to collect, structure and compare application design approaches as design patterns.**

Keywords- ubiquitous computing; pervasive computing; application design; classification; pattern language

I. Introduction

The research field of *ubiquitous computing*, also referred to as "ubicomp", was founded by Mark Weiser who presented the concept's idea in his work "The Computer of the 21st Century" [1]. Smart devices that are equipped with sensors or that are capable of providing information silently integrate into the environment. The communication between different entities should ideally take place in a wireless manner. This way, a number of smart devices together shape a *ubiquitous computing environment*.

The current situation where a personal computer drags all the attention towards itself should completely be avoided such that users are able to concentrate on the tasks they wish to perform instead of caring of the interaction. Computations can be performed inside the smart devices themselves or performed on machines inside the room. Connection and tasks management must not be the user's concern.

Weiser compares his idea to the ancient art of writing. Nowadays, we consume and provide information by simply reading or writing it - we are making use of this technique although we do not mandatorily need to know how to produce ink or paper, for example.

The concept of working with technology without having to know much about the details of the underlying infrastructure is the core idea of ubiquitous computing. Working also means using or even living in ubiquitous computing environments.

Currently, devices that can be used in a very "ubiquitous" way are entering the market; mainly these are netbooks and smartphones. This class of devices are first candidates to make ubiquitous computing widely available since there is a still growing increase in sales numbers as stated by Gartner [2]. With a high degree of connectivity and new generations of different kinds of sensors, new applications and ways of interaction become possible that were still visionary some years ago.

A. Functionality Everywhere

When talking about different service networks and the provisioning of services, there is also the need for looking beyond the personal (and limited) scope of mobile devices. Following the concepts of *Pervasive Computing*, the *Internet of Things (IoT)* or *Cloud Computing*, there are far more possibilities to offer services, since:

- The concept of *Pervasive Computing* allows the integration of computing power into real world objects, devices and environments [3].
- *Labeling* and therewith the IoT concept gives the possibility to uniquely identify and address real world objects [4]. Thus, the possibility of potentially unlimited labeling holds chances and challenges for many different kinds of applications as outlined by [5].
- In case that resources are too weak or cannot fulfill the requirements of task, these tasks are outsourced into the *Cloud* and thus virtually extend the devices' resources and transform them into gateways accessing more powerful functionalities [6].
- The growing *network infrastructure* allows communication between devices and thus the exchange of information or the consumption of services. An overview of the mobile phones and network infrastructure generations can be found online at [7].

ISBN: 978-1-61208-111-3

Services that are deployed together with real world objects and which are accessible via any kind of network are called *smart services* for the rest of this document.

B. *Realizations*

The above concepts and visions are partially already a reality. Amazon or Google, for example, provide access to their processing powers by introducing the Amazon Elastic Compute Cloud [8] or the Google App Engine [9].

Additionally, a combination of using GPS data together with permanent network access by mobile providers are used in projects like Layars [10] or Wikitude [11]. These mobile approaches make use of the mobile device's position and access databases over the mobile network in order to present additional information about objects next to the current position. These approaches are based on the Magic Lens approach first introduced by Bier et al. [12].

Later work incorporates mobile projectors to augment physical objects like paper maps and project information directly onto the object [13].

Further ideas than only receiving information are applicable which is for example already realized by the UbiLens project at Fraunhofer FIT [14]. In this project, smart services attached to different kinds of real world objects can be consumed, ranging from information retrieval over triggering actions up to the combination of different services and devices. An online available example shows how different real world objects are recognized by a server in the background receiving the camera image from the mobile phone. After identification of the object, different services can be selected on the mobile phone according to the purpose of the real world object [15].

II. Problem and Approach

A manifold of research approaches explore ways to offe ubiquitous functionality in public and private environments. Since each approach concentrates on different scenarios, uses different hardware and calls its components differently, it is hard to find similarities within existing implementations.

We see the danger of the repetition of design failures, unused chances of extending successful designs and missed chances to learn from realized approaches. These dangers may result in loss of design time and even money.

We consider a dynamic pattern language extracting and structuring results from different approaches as a possible solution to that problem. The pattern language will help new application designers in the field to more easily find working design approaches and modify them to their needs. This will also support the knowledge management within research communities and enterprises. Domain novices can pick up expert knowledge gathered from experience and formulated in the pattern language.

The rest of the paper is structured as follows: Section III provides an overview of different pattern languages used in different domains. In the preceding section (cf. section IV), we discuss features that are, from our point of view, missing in the current approaches of pattern languages. In section V, the idea of the common classification scheme is shaped addressing one of the problems discussed. The last section gives an overview of the planned next steps towards the intended pattern language and its intended new features (cf. section VI).

III. Pattern Languages

There is a variety of application domains and the popularity of gathering knowledge in pattern languages that are more or less technically formulated.

In the domain of HCI design patterns, Borchers, Schümmer and Stephan Lukosch follow the basic structure of Alexandrian design patterns (cf. [16]) by making use of natural language in order to describe solutions to specific design problems [17], [18]. In their approach, they structure a pattern into the following parts which are often similarly adapted in other pattern languages:

A *name*, *sensitizing picture*, the *intent* summarizing the pattern's solution in one sentence, the *context* in which the problem occurs and the solution is described, a *problem* description containing the most important aspects, a *scenario* putting the pattern's problem into an illustrating example context in order to increase understandability, *symptoms* helping the reading to find out about conflicting forces within the context, a *solution* to the conflicting forces problem, *dynamics* naming actors and components involved in the pattern, *rationale* providing explanations for a pattern's success and applicability, *checks* that pose questions that try to help the reader to figure out whether the pattern representing a template solution was well adapted to the current design problem, *danger spots* showing potential new problems that may occur when applying the pattern.

So they can be regarded as warning features trying to avoid the blind application of a pattern. The closing sections of a pattern are named *known uses* representing the second part of a pattern's ”proof” by presenting approaches in which the pattern is successfully applied and *related patterns* linking to relevant alternatives, patterns that are important for other stakeholders or patterns that go into more detail of a possible solution.

Pattern languages are also to be found in different application domains reaching from technical software design in object-oriented programming (cf. [19]) to interface design up to organizational patterns in business structures.

Rising and Manns, for example, present ways to restructure existing organizational structures and to introduce new ideas into an existing system. Their pattern language *Fearless Change* relies on social structures and requires practices that establish trust in new goals [20].

The *Organizational Patterns* language concentrates on team interaction in software projects as described by [21].

Another example is the *TELL* project that present patterns for computer-supported learning [22].

Teachers are supported in questions about group-based learning with collaboration technology find advice in the *Pedagogical Patterns Project* as described by [23] and in the domain of business process management, the approach by [24] represents an attempt to describe socio-technical systems following strict business processes. The pattern language developed is called *Workflow Patterns*.

The domain of user interface design is also served by the pattern languages *Web Usability Features* ([25]), *Web Patterns* [26] and for example the *Amsterdam Pattern Collection* [27]. Tidwell presents a very comprehensive pattern language for computer-mediated interaction in the context of non-web-based applications [28].

In many different fields where knowledge and experience can be captured interlinked, pattern languages have proven to be a useful approach.

IV. Missing Features

Current pattern language approaches provide a good structural basis for capturing design knowledge for specific application domains mostly explored by a small group. However, concerning the idea of covering design guidelines from many different projects and groups, we consider more features for a pattern language extending the current concepts as needed. These features are described in the following.

A. Lacking Application Domain Independence

Current pattern language approaches are mostly bound to one specific application domain and a small set of involved group. From the given circumstances and denominations, patterns are created and arranged in a pattern language.

The intended pattern language is intended to cover many different research groups and commercial projects, where different ideas and approaches are implemented and evaluated.

Many approaches are situated in the field of ubiquitous computing but the *naming* of components, techniques and concepts differs widely. A comparable classification and naming scheme is needed in order to be able to generalize concepts and interconnect common knowledge as well as to search within existing work results.

This is, in our opinion, a needed requirement to be able to integrate different approaches in different application domains.

B. Lacking Extensibility and Openness of Knowledge

Even after extracting knowledge about a certain aspect of ubiquitous computing interaction design, it is hard to discuss results from existing and new approaches. Once published, they remain within the documents and need to be refined or discussed besides the actual publication in follow-up research, forums or conferences.

That makes the reuse and refinement of results a very hard task. At the moment, there are only limited ways to extend and discuss research results. Openness is only given in a passive way; the results can be read but not actively be extended or discussed.

C. Lacking of Recommendation

Considering patterns about applied concepts and techniques within the domain, there is no direct connection between them. The exchange of knowledge about combinations that were implemented and worked out well and those which did not is not given. Recommendations for proven combinations of concepts are missing and therefore hindering the reuse and extension of concepts.

Again, time-to-market and time-to-research-results can be shortened by providing suggestions for good combinations of smaller units of solutions from different approaches.

D. Lacking of Knowledge about "Bad Practices"

In publications, often results reveal information about working concepts that were successfully implemented. Only initial studies about a certain problem domain concretely outline deficits in order to justify and motivate intended research.

However, in a domain that is actively being explored, failures or methods that were not accepted are not always clearly described or even mentioned. In our opinion, this is often the case in research about interaction techniques and metaphors. Here, authors mainly describe working solutions and drawbacks are omitted.

We consider the inclusion of bad practices that are not trivial and were revealed unexpectedly in experiments and implementation a very important feature to be integrated into the pattern structure.

V. A Classification Scheme for Ubiquitous Computing Environments

As a first step towards a pattern language of application designs, a common denomination for similar approaches is needed. Once different approaches can be described by a common vocabulary, the inherent design knowledge can be compared more easily, discussed and transferred to different application scenarios.

In ubiquitous computing environments, there are different objects providing functionality and interoperating among each other, the environmental infrastructure and the user. Different kinds of functionality is identified and abstracted to a semantic level therefore called *smart service* (cf. Section V-B). Real world objects that are augmented with smart services are referred to as *smart objects* (cf. Section V-A). Smart objects augmented with an arbitrary number of smart services together build a *smart environment* within an application domain (cf. Section V-D) .

The definitions given in this work will constitute the foundation of the intended pattern language approach and cover already existing applications that are found within current research and projects but will also be use to describe future ideas and approaches.

A. Smart Objects

In the scope of this work, a *smart object* can be any kind of device of object that provides *functionalities* augmenting its original purpose. This could be the provision of information related to the object or something more abstract it stands for. For example, a standard paper timetable in train stations could provide the same information that is printed on it in a virtual way. Furthermore, it could also stand for advanced routing in such a way that travelers use it to decide for a route from the current station to their destination. This routing could also be offered as a computer-supported services like it can be found on current websites driven by public transportation companies.

The idea is now to be able to attach any kind of service to a standard object or device which ideally are related to the object's original purpose or more abstract concepts it stands for.

Thus, in the scope of this work, any kind of object potentially provides an arbitrary number of smart services that provide information or extended functionalities. Consequently, every object that is augmented this way is referred to as a smart object.

Fig. 1 illustrates this concept showing arbitrary functionality, i.e., smart services, being connected to a real world object.

B. Smart Services

Smart services are virtually attached to physical objects or devices and therefore augmenting them with virtual functionalities. The functionality they provide could be informative or offer more sophisticated applications. Ideally, the services are related to the object's original purpose but theoretically, they could offer anything developers have in mind.

However, the applicability and user's understanding would definitely suffer from such implementations since they would not really convey a coherent meaning like in the timetable example given in Section V-A.

Currently, the derived classification scheme covers three different kind of smart services. They can

- provide information (*provider*)
- provide ways of interaction (*connector*)
- process input (*consumer*)

Each kind of smart service optionally possesses a special attribute which is called the *takeaway - attribute*. This optional specialization of a service enables the user to take the offered functionality with her so that it still available later and when she is not necessarily close to the smart object. A service which does not posses the takeaway-attribute can only be used within direct contact with the smart object.

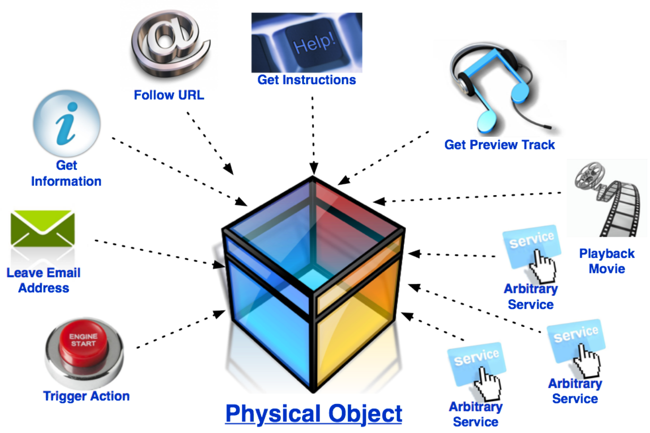


Figure 1. A standard real world object can be augmented by an arbitrary number of different kinds of smart services. The functionality can be freely defined.

In order to illustrate the classification scheme, one small example per kind of service is given in the following. Additionally, a possible implementation and usage of the takeaway-attribute is explained.

(i) *Providers* are implemented as services that offer certain information about a special location like point of interest (POIs). The Wikitude project is one example of an application that provides information bound to certain buildings or locations. Google Goggles (cf. [29]) also applies this approach by connecting information to buildings.

(ii) *Connectors* enable the user to control an offered service. One example is a public display, that offers a smart service allowing users to remotely control the display. That way, the user's input is processed and directly fed into an application lying behind the service. Also, the timetable scenario mentioned in Section V-A can be implemented in an interactive way such that the user influences parameters like price, travel duration or comfort class when planning his connection.

(iii) *Consumer services* do not necessarily provide sophisticated feedback to the user or provide a direct result. They are moreover be regarded as triggers for certain processes or applications. Examples for this are push-services that for examples upload just-taken pictures to an online community account like Facebook.

Another example is represented by macros like those used in the home automation sector. A service called ”turn on the lights” is then used as a trigger. This way, the user's input is consumed and a whole system, the home automation system in this case, takes over.

C. The Optional Takeaway-Attribute

As an optional add-on, the takeaway attribute comes into play. In the context of the timetable example *providing* information about connections, this means that the train arrival and departure times can be taken away by the user and therefore are also be available when she is not standing next to the timetable. That way, information can be collected and consumed later and repeatedly.

For *connectors*, it makes sense to use an application connected to an interactive service also at a later time. The can make a train booking at home but have the virtual service still available for tracking the booking process or eventual changes. So users are able to connect a real world object to a *virtual pendant* offering similar or extended functionalities.

Consuming services can also offer the takeaway-attribute. In the example of home automation, the user is then able to turn on the lights while she is still outside and does not first have to search for the service in the dark.

Fig. 2 shows some examples of service-augmented real world objects, optionally with the take-away attribute. In the example, an ordinary timetable at a train station is augmented with service finding the train connection from the current location to the desired destination. This service posses the takeaway-attribute since it could also be used again at a different location even if the user is not near the real world object.

The TV screen as an ambient display offers the service to provide video content that can be played on the screen but also played back on te the user's mobile device. Another service combined with ambient speakers offered is the playback of user-provided content that is only available when she is present at the screen. Remote control is not allowed here.

In the case of the coffee-machine, the takeaway attribute is not given since this services of filling a cup or retrieving information about the coffee status is only available when a user is directly interacting with it due to security reasons.

These examples show that the takeaway-attribute needs to be used carefully in order to provide meaningful services. Services that provide physical feedback like printouts (e.g., tickets) or products (e.g., coffee from an augmented coffee machine) need to ensure that the user is directly available.

D. Smart Environments

A setup with arbitrary kinds of services attached to an arbitrary number of real world objects, is referred to as *smart environment*. Different classes of smart environments are possible. Based on the purpose of the services (entertainment, technical support, maintenance) or the location of the environment and the level of publicity or privacy, respectively. Examples for these kinds of environments are public spaces like train stations, official buildings, airports or sights in a city. More private or security related situation can be found at home or in office spaces.

Applications like Wikitude (cf. [11]) or Layars (cf. [10]) already turn public spaces into a kind of "informational smart environment" by displaying additional information about a location the user is close to.

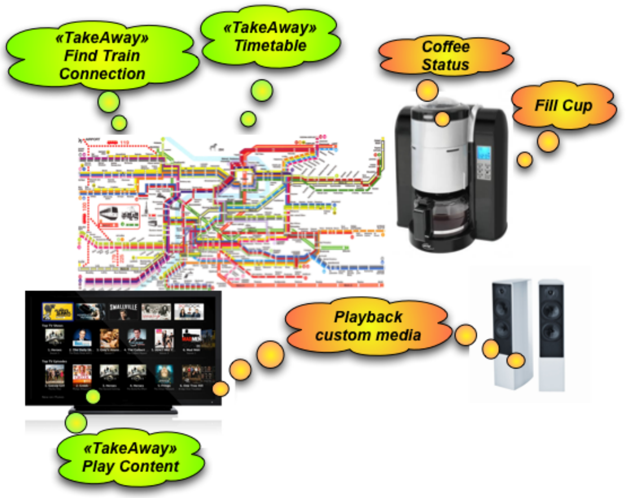


Figure 2. In this example, services with the takeaway-attribute are connected to a metro plan and a TV screens. The coffee-machine and speakers providing playback services but only allow direct interaction.

VI. Future Work

The long-term intention of our work is to develop a pattern language for application design in ubiquitous computing environments. As a first requirement for the creation and extension of such a structure we consider a *common vocabulary* as a necessary requirement for comparing different approaches and to extract knowledge from them. The extraction will be presented in form of design patterns, similar to approaches presented by Schümmer and Lukosch, Borchers and Gamma et al. (cf. [18][17][19]). The latter work addresses the technical part of application design in terms of how a technical problem can be solved by applying software patterns. The former patterns are formulated against specific application domains, i.e., CSCW and HCI, that primarily describe design knowledge of applications on a conceptual level. Technical suggestions play a minor role. Like Borchers, Schümmer and Lukosch, the patterns for ubicomp application design are intended to be arranged in a pattern language and therefore interconnecting patterns. The deeper readers follow the structure, the more details of the application design are described.

New features like decision nodes separate different kinds of interactions depending on the usage scenario. Recommendation mechanisms will help to find successful combinations of patterns. Finally, the pattern language is intended to be open to new patterns that can be integrated.

The design patterns will range from the discovery of smart services, over the interaction until user preferences and privacy and security patterns.

VII. Conclusion

This work presents first conceptual steps in the progress of finding a classification scheme that is able to map denominations from existing and future approaches to a generally applicable vocabulary needed for the formulation of design patterns in the domain of ubicomp application design. Next, existing approaches in the domain of mobile applications in ubicomp environments will be analyzed and the presented scheme will be mapped to these approaches. After translating the approaches to the abstract vocabulary, patterns of successfully implemented concepts supported by published evaluations will be derived and discussed within the community. From the initial set of patterns, the other requirements described in section (II) will be addressed.

In a later step, new domains and ideally a large set of domains are to be analyzed. The results will either support the generality of the developed pattern language and its new features or confute the approach. From our point of view, the assembly of patterns in a pattern language will support the gathering, structuring and extraction of design knowledge from current and future approaches and make them comparable and thus facilitate discussion, reuse and modification throughout application scenarios.

References

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] "Gartner newsroom - press release, may 19th, 2010," 2010, http://www.gartner.com/it/page.jsp?id=1372013.

[3] U. Hansmann, L. Merk, and M. S. Nicklous, *Pervasive Computing - The Mobile World*. Berlin: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2001.

[4] H. Chaouchi, *The Internet of Things: Connecting Objects*. ohn Wiley & Sons, 2010.

[5] C. Floerkemeier, M. Langheinrich, E. Fleisch, and F. Mattern, *The Internet of Things: First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008, Proceedings*, 1st ed. Springer-Verlag Gmbh, 2008.

[6] J. Rhoton, *Cloud Computing Explained: Implementation Handbook for Enterprises*, 2nd ed. Recursive Press, 2010.

[7] C. M. S. Ltd., "Telecoms market research," website, 2008, http://www.telecomsmarketresearch.com/resources/Mobile_Phone_Market.shtml.

[8] "Amazon elastic compute cloud," 2010, http://aws.amazon.com/ec2.

[9] "Google app engine," 2010, http://code.google.com/intl/appengine/appengine.

[10] "The layars project," 2010, http://layars.com.

[11] "The wikitude project," 2010, http://wikitude.org.

[12] E. A. Bier, K. Fishkin, K. Pier, and M. C. Stone, "Toolglass and magic lenses: the seethrough interface," *Proceedings of SIGGRAPH*, vol. 93pp, pp. 73–80, 1993.

[13] J. Schöning, M. Rohs, and S. Kratz, "Map Torchlight: A Mobile Augmented Reality Camera Projector Unit," *Information Systems*, 2009.

[14] V. N. Wibowo, "The UbiLens Approach - Visualisation of and Interaction with Real World Objects through a Moble Phone's Camera ," master thesis, Fraunhofer FIT, 2010.

[15] "Gartner newsroom - press release may 19th, 2010," 2010, http://www.youtube.com/watch?v=IY1FmKhfAao.

[16] C. Alexander, *A Pattern Language: Towns, Buildings, Construction*. New York, New York, USA: Oxford University Press, 1977.

[17] J. Borchers, *A Pattern Approach to Interaction Design*, 1st ed. John Wiley & Sons, 2001.

[18] T. Schümmer and S. Lukosch, *Patterns for Computer-Mediated Interaction*. Chistester, West Sussex, England: John Wiley & Sons, 2007.

[19] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*, 1st ed. Amsterdam: Addison-Wesley Longman, 1995.

[20] L. Rising and M. L. Manns, *Fearless Change: Patterns for Introducing New Ideas: Introducing Patterns into Organizations*, 2005th ed. Amsterdam: Addison-Wesley Longman, 2005.

[21] J. O. Coplien and N. B. Harrison, *Organizational Patterns of Agile Software Development*, illustrated ed. Prentice Hall International, 2004.

[22] F. L. National Centre and Literacy, "The Tell Project." [Online]. Available: http://www.tell.praesa.org/

[23] J. Eckstein and J. Bergin, "The Pedagogical Patterns Project," 1999. [Online]. Available: http://www.pedagogicalpatterns.org/

[24] C. Hentrich, "Six patterns for process-driven architectures," in *Proceedings of the 9th Conference on Pattern Languages of Programs (EuroPLoP 2004)*, 2004.

[25] I. Graham, *A Pattern Language for Web Usability*. Amsterdam: Addison-Wesley Longman, 2003.

[26] D. Schwabe and G. Rossi, "The object-oriented hypermedia design model," *Communications of the ACM*, vol. 38, no. 8, pp. 45–46, August 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=208344.208354

[27] M. van Welie, "The Amsterdam Pattern Collection," 2010. [Online]. Available: http://visiblearea.com/cgi-bin/twiki/view/Patterns/Amsterdam_Collection_of_Interaction_Design_Patterns

[28] J. Tidwell, *Designing Interfaces*, 1st ed. O'Reilly Media, 2005.

[29] "Google goggles," 2010, http://googlegoggles.com.

Detection of Generic Micro-architectures on Models

Cédric Bouhours, Hervé Leblanc, Christian Percebois, Thierry Millan
IRIT – MACAO team – University of Paul Sabatier
118 Route de Narbonne
31062 TOULOUSE CEDEX 9 FRANCE
{bouhours, leblanc, percebois, millan}@irit.fr

***Abstract*— Existing pattern detection methods generally use code information obtained during reengineering process. However, none of these methods exclusively works with design information. In this paper, we propose a novel pattern detection method based on structural properties of UML models. This technique allows the detection of any kind of generic micro-architecture, like design patterns or spoiled patterns. Since a generic architecture is context-free, the structure of the searched fragments depends on the use context. So, our technique uses a structural concordance paradigm to identify all possible instantiations of a generic micro-architecture. To increase the precision of the detection, authorized, prohibited, and optional relations can be directly precised into the micro-architecture model.**

Keywords- Pattern detection. Graph isomorphism. UML Model.

I. INTRODUCTION

Various works aim at identifying fragments representing correct, incorrect or incomplete instantiations of design patterns, in order to help the comprehension of existing designs and to provide a base for possible improvements [1]. To identify characteristic fragments, it is necessary to parse models, to ensure that the execution time of the algorithm is adapted to consequent models and to recognize a form that is approximate or to supplement. This approximation is very problematic, because it introduces uncertainties into the research. In the case of design patterns, the designer adapts the pattern to his problem, obliging the detection methods to be able to detect every possible form [2]. To render possible these detections, some tools use source code to identify complete or distorted versions of design patterns [3]. The information extracted from the source code augments the precision of fragment intent and so the pattern detection.

However, during model-driven processes, the identification of patterns concerns the designer in order to target specific model fragments. For example, spoiled patterns allow the detection of fragments substitutable with design patterns before a coding stage [4]. Thus, we have conceived a detection method based on UML structural properties of UML models. Thanks to this method, we are able to identify instantiations of generic micro-architectures, like design patterns or spoiled patterns. The first intent of our detection method concerns the detection of spoiled pattern, which we present in Section 2. Section 3 presents the model representation we use to formalize our detection technique, and takes a stand on our work in relation to existing graph matching problems. The remainder of the paper is composed by the techniques used to compute the detection (Section 4), and some validation tests in Section 5. The paper ends with a discussion of related works and a conclusion, in Sections 6 and 7. The main contributions of this paper are the specification and the implementation of a generic UML graph matching method able to detect pattern instantiations whatever their form.

II. SPOILED PATTERN DETECTION

Choosing a good design pattern and ensuring the correct integration of the chosen pattern are non trivial for a designer who wants to use them. To help designers, we propose design inspection in order to detect “bad smells in design” and models reworking through use of design patterns. The automatic detection and the explanation of the misconceptions are performed thanks to spoiled patterns [4].

If we consider that a design pattern is the optimal reusable micro-architecture for a type of problem, then for each design problem that is solvable with a design pattern, the optimal solution is the instantiation of the design pattern. Moreover, if we consider an alternative solution as a valid solution but with a different architecture compared to the optimal solution, then, an alternative solution is an inadequate solution for a given problem, and is substitutable with the instantiation of the concerned pattern. A spoiled *Composite* pattern is given in Fig. 1.

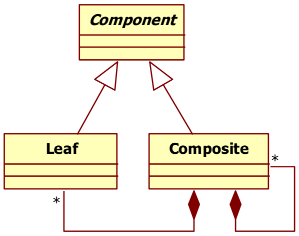


Figure 1. A spoiled pattern (development of the composition on *Composite*)

Each spoiled pattern has a name that describes the misconception: here the development of the composition link on the composite participant of the pattern. So, there is not a maximal factorization of the composition which implies addition or removal of a leaf or a composite need code modification.

Structurally, a spoiled pattern is represented at the same level of granularity as a design pattern allowing us to identify them as design patterns. An alternative fragment is a model fragment such as its structural properties match with the structural properties of a spoiled pattern and whose intent conforms to the corresponding pattern. Then, after the detection, an alternative fragment can be considered as potential. The validation of its intent is assumed by the designer of the user model. More details on these concepts, especially the collect, the catalog, the use in a tooled design review activity, and the refactorings can be found in [12].

The static UML model in Fig. 2 represents a basic architecture for a file system management. Authors of this model are interested in the presentation of some object concepts: *inheritance between classes* (a uniform protocol for every *FileSystemElement* is encapsulated by a corresponding abstract class) and *management of reference and delegation* (there are composition links between container and components).

Nevertheless, this model contains a misconception. Although there is a uniform protocol owned by the class *FileSystemElement*, the composite links management along a hierarchical structure is duplicated. Indeed, the *Directory* class manages independently links on *Files* and *Directories*. {*Directory*, *File*, *FileSystemElement*} is an exact instantiation of the *Composite* spoiled pattern. It is easy for the designer to see that this fragment has the same intent as the *Composite* pattern and to consider it as a bad smell in design. Furthermore, when the authors have implemented this model, they realized that there were defects. They adapted their code to correct them, without changing the design model. Therefore, the fragment must be substituted with the instantiation of the composite pattern on the user model or context.

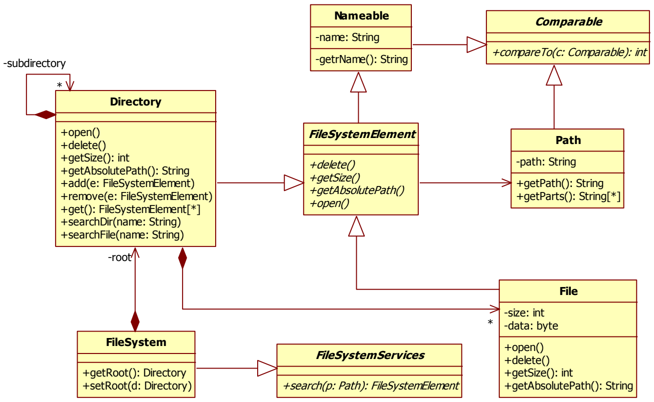


Figure 2. A File System Management Design

During a process development, it is more interesting to detect bad smells in design before the coding stage. Indeed, the model correction is easier and uses less time if the code is not already written. So, we have conceived a detection technique working with design information, and without information issued from reverse engineering process. The existing techniques use code information issued from their own reversion methods.

As a spoiled pattern has the same abstraction level as a design pattern, we consider that they are both "pattern" and so "generic micro-architecture". Our detection technique is able to detect generic micro-architecture, and so, the remainder of this paper uses "pattern" term to mean "design pattern", "spoiled pattern" or "generic micro-architecture". For the sake of clarity, we use the *Composite* design pattern as an example.

III. Graph Representation

We consider models described in UML 1.5 [5] according to the XMI standard [6]. With this meta-model, models can be represented by directed graphs. A graph consists of typed nodes representing the classes and the relations between them. Arrows are used to indicate the direction of the relations between classes. In our case, we are interested by classes, associations, and generalizations only. There is a gap between the visualization and the internal representation of a UML model.

Fig. 3 illustrates these two representations in UML 1.5 for a design pattern: in a class diagram and in a graph conforms to XMI format. In this example, the design pattern is a simple directed graph, with the vertices C, L and Co, respectively *Component*, *Leaf* and *Composite*. There are also sets of vertices {A}, {AE}, {G} and {S}, respectively *Association*, *AssociationEnd*, *Generalization* and *Specialization* of the UML meta-model.

We have separated the vertices in two different subsets: ***Vc*** containing all the classes of the model, and ***Vm*** containing all the meta-classes allowing the connections of the classes. The vertices of the set {AE}, for *AssociationEnd*, come from the meta-model and are used to connect *Classifier* to *Association*. These vertices are tagged by *AssociationEnd* meta-class attributes in order to characterize the extremity of associations. For example, for the vertex Co, the adjacent vertices are AE, G and S only, excluding A which is accessible from AE only.

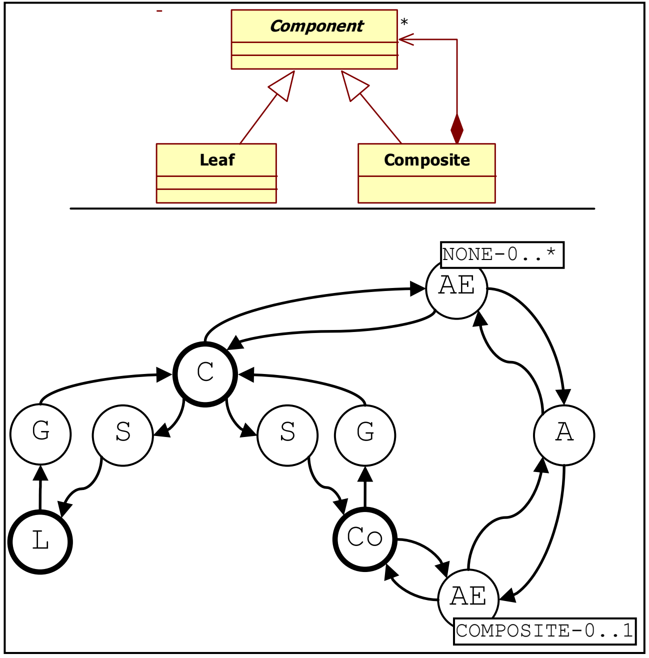


Figure 3. A UML model and its directed graph representation

As we consider UML models as graphs, we can formulate our patterns identification problem as a problem of sub-graphs or directed sub-graphs identification in a graph. There are two main approaches in this domain. The first one is known as exact graph matching, which consists in finding exactly a given subgraph in a graph [7]. The second approach consists in identifying all the sub-graphs looking like more or less a given graph [1] and is called inexact graph matching.

Exact graph matching algorithms require the examination of all possible sub-graphs that have the same number of nodes and arrows with the source graph, which means a NP-complete problem [7]. For our problem, exact pattern matching algorithms are inefficient. As a pattern is a generative base of a family of specific designs, we do not exactly search the generic micro-architecture, but one of its instantiations, which are not known in advance.

Inexact graph matching algorithms are very useful when an isomorphism between two graphs cannot be found or is too strict for research. They find the best correspondence between two graphs. For example, some algorithms calculate the distance between two graphs, expressed for instance in number of modifications to transform a graph to the compared graph [1]. In the context of pattern detection, such algorithms are more interesting, because they are able to detect sub-graphs structurally close to the pattern. However, it is not sufficient, because a given design pattern may have several forms depending on the instantiation context.

Since we cannot use exact or inexact pattern matching, we have defined a detection method working by structural concordances. Thanks to structural properties allowing the structural detection of pattern, this technique is able to detect pattern instantiations, whatever their form, and taking into consideration authorized, prohibited or optional relations between classes, as described in section 3.3.

IV. Specification of the Detection

A pattern is described with a set of structural properties allowing its structural description, and thus the detection of its instantiation in a model. We have decomposed the remarkable properties into two subsets: the local properties that characterize individually each class and the global properties which characterize the classes against each other depending on their inter-relations. This separation allows us to constitute different filters during the detection, through use of structural similarity comparisons. The result of the search is a set of fragments identified in the model analyzed.

A. Structural Concordance

The structural properties of a pattern enable us to detect fragments in models. Compared to graphs, they enable us to detect sub-graphs families, because they describe the patterns as well as the fragments they can generate. Our detection method uses the local and global structural properties to check the structural concordance of the fragments with the patterns.

Definition 1 presents in a formal way a model ***m***. As seen previously, it is a directed graph with two sets of vertices ***Vc*** and ***Vm***, respectively representing the model classes and the instances of meta-classes describing the relations between the classes.

$$
\begin{aligned}
&\forall m \in \{Model\ to\ analyse\}: \\
&\quad m = directed\ graph(\boldsymbol{Vc}_m, \boldsymbol{Vm}_m, \boldsymbol{E}_m)\ / \\
&\quad \boldsymbol{Vc}_m = \{Model\ classes\} \\
&\qquad \text{such as x} \in \boldsymbol{Vc}_m \Rightarrow \text{x instance\_of Classifier} \\
&\quad \wedge \boldsymbol{Vm}_m = \{Relations\ between\ classes\} \\
&\qquad \text{such as x} \in \boldsymbol{Vm}_m \Rightarrow \text{x}\ instance\_of \\
&\qquad\quad \text{AssociationEnd} \vee \text{Association} \\
&\qquad\quad \vee\ \text{Generalization} \vee \text{Specialization} \\
&\quad \wedge \boldsymbol{E}_m = \{Directed\ Edges\} \\
&\quad \wedge \forall c \in Vc_m, adjacent(c) \subseteq \boldsymbol{Vm}_m
\end{aligned}
\qquad (1)
$$

Like for a model, we formally define a pattern in definition 2.

Each pattern has a unique reference participant which we note ***reference_dp***. It represents a particular vertex of ***Vc_dp*** that we detail in part B. This vertex is chosen by an oracle according to its structural complexity and its responsibilities on the problem to solve.

$$
\begin{aligned}
&\forall dp \in \{Pattern\}: \\
&\quad dp = directed\ graph(\boldsymbol{Vc}_{dp}, \boldsymbol{Vm}_{dp}, \boldsymbol{E}_{dp}) \\
&\qquad\ \text{and reference\_dp}\ / \\
&\quad \boldsymbol{Vc}_{dp} = \{Pattern\ participants\} \\
&\qquad \text{such as x} \in \boldsymbol{Vc}_{dp} \Rightarrow \text{x}\ instance\_of\ \text{Classifier} \\
&\quad \wedge \boldsymbol{Vm}_{dp} = \{Relations\ between\ participants\} \\
&\qquad \text{such as x} \in \boldsymbol{Vm}_{dp} \Rightarrow \text{x}\ instance\_of \\
&\qquad\quad \text{AssociationEnd} \vee \text{Association} \\
&\qquad\quad \vee\ \text{Generalization} \vee \text{Specialization} \\
&\quad \wedge \boldsymbol{E}_{dp} = \{Directed\ Edges\} \\
&\quad \wedge \forall c \in \boldsymbol{Vc}_{dp},\ adjacent(c) \subseteq \boldsymbol{Vm}_{dp} \\
&\quad \wedge reference\_dp \in \boldsymbol{Vc}_{dp}
\end{aligned}
\qquad (2)
$$

Thus, we have two directed graphs where we search for combinations of occurrences of the first in the second. In order to avoid a combinatorial explosion of the research possibilities, and thus to limit the problem complexity, we do a first filtering of the sets of the vertices having the adequate local properties.

The first step consists in searching for all the vertices of graph ***m*** in accordance with the predicate 5 ***local_SPC***. This predicate, meaning "structural properties concordance", allows to check if a vertex ***c*** of the graph ***m*** has, at least, the same adjacent vertices as a vertex ***p*** of ***dp***. Thus, if ***c*** is ***local_SPC*** with ***p***, the class corresponding to ***c*** has, at least, the same local structural properties as the participant of the pattern corresponding to the vertex ***p***. The comparison between the adjacent vertices is done with an equivalence relation comparing the type and the attributes of the adjacent vertices, as definition 3 shows it, with a constraints relaxation presented in part 3.3.

$$
\begin{aligned}
&\forall (e1, e2) \in \boldsymbol{Vm}_m \times \boldsymbol{Vm}_{dp} : e1 \cong e2 \\
&\qquad \Leftrightarrow type(e1) = type(e2) \\
&\qquad \wedge\ \forall\ attr \in \{\text{attributes of } e1 \text{ and } e2\}: \\
&\qquad\qquad valuation(attr_{e1}) = valuation(attr_{e2})
\end{aligned}
\qquad (3)
$$

By extension, we obtain the definition 4.

$$
\begin{aligned}
&\forall Ei \subseteq \boldsymbol{Vm}_m, \forall Ej \subseteq \boldsymbol{Vm}_{sp}: Ei\ \supseteq_{\cong} Ej \\
&\qquad\qquad \Leftrightarrow\ \forall ej \in Ej, \exists ei \in Ei\ /\ ej \Leftrightarrow ei
\end{aligned}
\qquad (4)
$$

As the adjacent vertices of a vertex of ***Vc*** belong to ***Vm*** and as all the vertices ***Vm*** are strongly typed, the vertices of ***Vc*** can be filtered thanks to their local properties, as predicate 5 shows it.

$$\forall c \in \boldsymbol{Vc}_m, \forall p \in \boldsymbol{Vc}_{dp}: local\_SPC(c,p) \Rightarrow adjacent(c) \supseteq_{\approx} adjacent(p) \qquad \textbf{(5)}$$

In order to illustrate the predicate ***local_SPC***, we search *Composite* design patterns in the model of Fig. 4, whose local properties of each participant are illustrated in Fig. 5. The result of the application of the predicate ***local_SPC*** is illustrated in Table 1. For more legibility, the models are represented in UML.

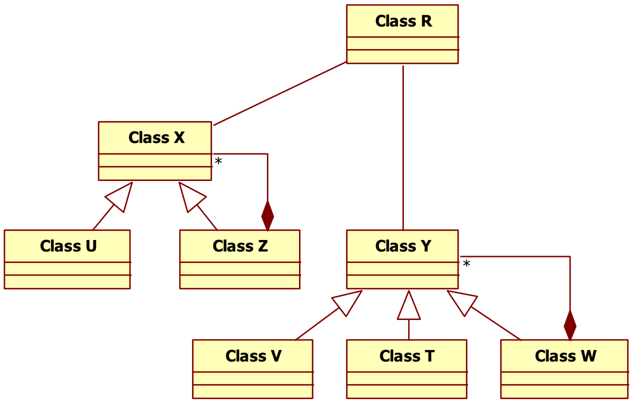


Figure 4. A model example

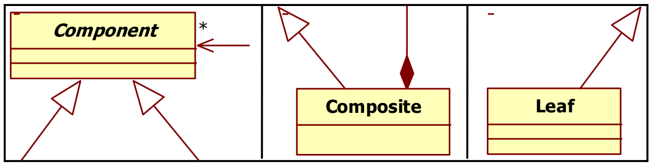


Figure 5. Local structural properties of the *Composite* design pattern

TABLE I. RESULT OF THE PREDICATE *LOCAL_SPC* ON THE MODEL

	Class							
local_SPC	R	T	U	V	W	X	Y	Z
Composite					OK			OK
Component						OK	OK	
Leaf		OK	OK	OK	OK			OK

The classes marked in Table 1 validate predicate ***local_SPC*** with the corresponding participant. It is possible to notice that the classes have the same adjacent vertices as their participants, except for classes *Y*, *W* and *Z* which have more. For example, we can note that the class *Y* has three daughters. It is partly thanks to the fact that a vertex of ***Vc*** can have more adjacencies, that we can detect all the various possible pattern instantiations. Moreover, we can notice that class *W* and class *Z* validate the predicate on two different participants from the pattern, *Leaf* and *Composite*. Indeed, the local properties of *Leaf* are included in those of *Composite*. Without the global properties, we cannot differentiate the *Composite* classes from the *Leaf* classes yet.

After comparing all the vertices of ***Vc***$_{dp}$ with all those of ***Vc***$_m$, i.e. all the participants of the pattern with all the classes of the model to analyze, we obtain a set of vertices having their adjacent vertices at least identical to those of the participants of the pattern. This first predicate is used as filter on the sets of the vertices of ***m***.

Predicate 6 ***global_SPC*** allows to check the concordance of the global properties, i.e. if a subgraph ***sf*** of ***m*** is isomorphic to ***dp***.

$$\forall sf, \forall dp : global\_SPC(sf, dp) \Leftrightarrow \exists g : isomorphism\ between\ sf \text{ and } dp \qquad \textbf{(6)}$$

A sub-fragment ***global_SPC*** with ***dp*** has, by definition, the same number of vertices as the pattern. Although the instantiation of the pattern causes the multiplication of some vertices, all the combinations, such as each class represents a distinct participant, remain isomorphic with the pattern. Fig. 6 illustrates this isomorphism of the sub-fragments of the model presented in Fig. 4.

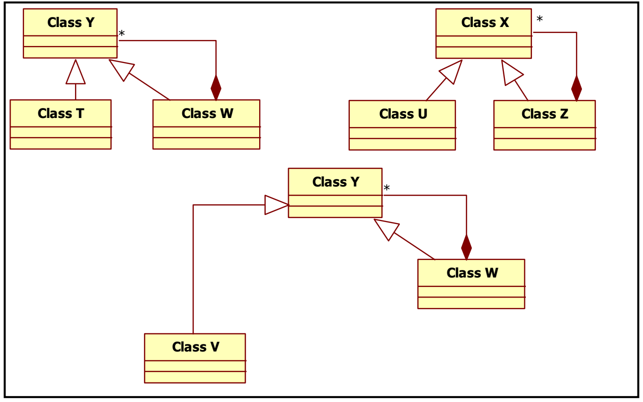


Figure 6. Sub-fragments of Fig.4 isomorphic with the pattern of Fig. 3

In the model of Fig. 4, according to the vertices identified as being ***local_SPC*** with the vertices of the pattern, we can build only three sub-fragments in conformity with ***global_SPC***. For example, the combination class *U*, class *W* and class *Y*, is not a sub-fragment, because even if there is the same number of vertices as in the pattern and each vertex is ***local_SPC*** with a vertex different from the pattern, there is no isomorphism between this combination and the pattern.

Thus, the predicate ***global_SPC*** enables us to eliminate class *W* and class *Z* from the *Leaf* responsibilities, since it is not possible to build a combination of classes in conformity with the predicate ***global_SPC*** with one of these classes to the responsibilities of *Leaf*.

Now, we have to build the complete fragments, i.e. to couple the sub-fragments which share the same vertices. A complete fragment ***cf*** is a subgraph of ***m*** including at least an isomorphic sub-fragment with the pattern and such as any graph induced by a combination of vertices referring once each participant of the pattern remains isomorphic with the pattern. Moreover, only one vertex of ***cf***, that we name ***reference_class***, is ***local_SPC*** with ***reference_sp***, the reference vertex of the pattern. In the case of Fig. 6, if we consider that *Composite* is the reference participant, we can regroup the sub-fragments {*ClassY*, *ClassT*, *ClassW*} and {*ClassY*, *ClassV*, *ClassW*} to form a complete fragment.

Thus, the predicate 7 ***complete_fragment*** allows to check if a fragment ***cf*** is a set of sub-fragments, each one isomorphic with ***dp***.

$$\begin{aligned} &\forall cf \subseteq m, \forall dp : complete\_fragment(cf, dp) \\ &\quad \Leftrightarrow \forall sf \subseteq cf : |sf| = |dp| \\ &\qquad \wedge \forall \mathrm{c}_1, \mathrm{c}_2 \in Vc_{sf}, \\ &\qquad\qquad participant(\mathrm{c}_1) \neq participant(\mathrm{c}_2) \\ &\qquad\qquad\qquad \Rightarrow global\_SPC(sf, dp) \end{aligned} \quad \textbf{(7)}$$

The application 8 ***participant*** associates to each vertex of the fragment ***cf***, a vertex of the pattern such as it is in local concordance and which it is connected in the same way to the reference participant.

```
application $participant$ :
    $\boldsymbol{Vc}_{cf} \rightarrow \boldsymbol{Vc}_{dp}$
    $\mathrm{c} \mapsto \mathrm{c}'$
    $participant(\mathrm{c}) = \mathrm{c}'$
    if $local\_CPR(\mathrm{c}, \mathrm{reference\_dp})$ then
      $\mathrm{c}' = \mathrm{reference\_dp}$ and $\mathrm{c}'$ is the reference class        (8)
    else
      $local\_SPC(\mathrm{c}, \mathrm{c}')$
      $\wedge\, global\_SPC(sg(\mathrm{c}, \mathrm{reference\_class}), sg(\mathrm{c}', \mathrm{reference\_dp}))$
          where $sg$ is the inducted graph by the set of the
          considered vertices
```

In proceeding like that, we can build fragments representing all the possible instantiations of the pattern. Indeed, even if it is not possible to anticipate which form has a complete fragment, it is sure, whatever its form, that it is composed of isomorphic sub-fragments to the pattern, since all the possible complete fragments will always have their classes connected in the same way to the respective participants of the pattern.

Fig. 7 presents two fragments with isomorphic sub-fragments to a pattern.

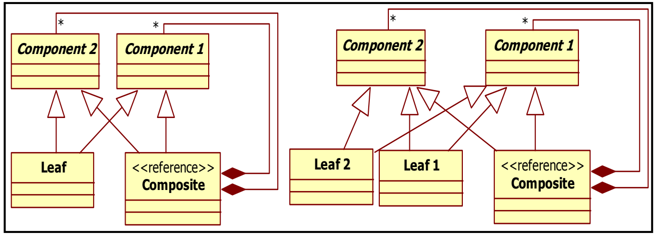


Figure 7. Two fragments with isomorphism sub-fragments to the *Composite* design pattern

The sub-fragments of the left fragment are {*Composite*, *Component1*, *Leaf*} and {*Composite*, *Component2*, *Leaf*}. Those of the right fragment are {*Composite*, *Component1*, *Leaf1*}, {*Composite*, *Component2*, *Leaf1*}, {*Composite*, *Component1*, *Leaf2*} and {*Composite*, *Component2*, *Leaf2*}. Thus, it is possible to notice that the two fragments presented are two different instantiations, but recognized as complete fragments.

A particular case is presented in Fig. 8.

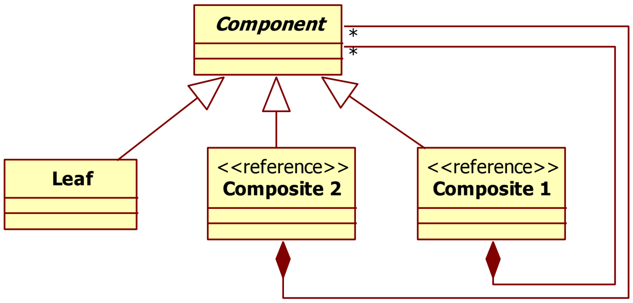


Figure 8. A particular case of a complete fragment

The two sub-fragments of this model are isomorphic to the *Composite* pattern: {*Composite1*, *Component*, *Leaf*}, {*Composite2*, *Component*, *Leaf*}. However, if we analyze this case, we can wonder whether, except its structure, it constitutes a true complete fragment. Indeed, the composite participant are not linked between them. Thus, we propose that Fig. 8 presents two distinct fragments {*Composite1*, *Component*, *Leaf*} and {*Composite2*, *Component*, *Leaf*}. We do not authorize a fragment to have two classes having the *Composite* responsibilities. We named this additional characteristic the "reference participant", necessary to the representation of the results of detection and the limitation of the matching complexity.

B. *Reference Participant*

Each participant of a pattern has not the same importance in the intent aimed by the pattern. The fact that a design pattern is the best solution resides in its structural organization, obligatory support with any collaboration between objects. For the *Composite* pattern, take into consideration the UML models presented in Fig. 9 and try to answer the question: can these models be still regarded as instantiations of the *Composite* pattern?

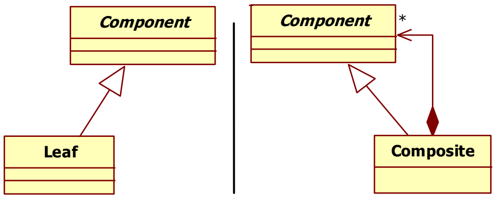


Figure 9. The *Composite* pattern without *Composite* and without *Leaf*

If we remove all the occurrences of the *Leaf* participant of the pattern, we do not loose the intent of the pattern, even if we lose the possibility of adding terminal elements in the hierarchical tree of composition. But, if we remove all the occurrences of the *Composite* participant, no more composition is possible. Indeed, it is *Composite* which completely manages the responsibilities for the composition of the objects, first intent of the pattern. Thus, this participant plays a dominant role in the pattern.

The reference participant depends on the pattern concerned. This participant is manually chosen with the heuristic evaluating its essentiality with the number of structural properties of each class.

ISBN: 978-1-61208-111-3

C. Authorized, Prohibited or Optional Relations

Any class not having the responsibilities of the reference participant can be found multiplied in the fragment, whereas those marked reference are separate in different fragments. But what does occur if a class of the fragment is connected of more than another way that envisaged? Fig. 10 illustrates this case on a fragment.

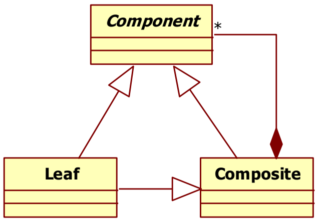


Figure 10. A model fragment with one supplementary relationship

If we apply the method introduced previously, the fragment is detected, because it respects strictly the structural properties of the *Composite* pattern. However, the additional inheritance modifies the responsibilities of the classes concerned. A *Leaf* inheriting a *Composite* does not have a meaning in the hierarchical composition of objects, since a terminal object cannot be a specialized non-terminal object. Thus, we discriminate any fragment having connections invalidating the intent of a pattern.

To recognize which relations of the fragment are discriminating, we documented the pattern with information indicating which connections are optional, obligatory or prohibited. This documentation is done thanks to a UML profile allowing us to add specific information on relationships. Fig. 11 illustrates the *Composite* profiled pattern.

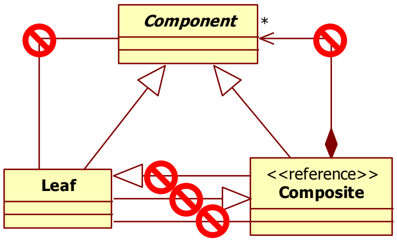


Figure 11. A graphical representation of the profiled pattern

Reported to the graph, we supplement the description of the pattern with a graph of the prohibited relations, comparable to the “Negative Conditions Application” (NAC) of graphs grammars [8]. If a connection appears in the graph of the pattern, it must obligatorily be present in the fragment; if it appears in the graph of the relations prohibited, it must absolutely miss in the fragment. If a connection is present in the fragment, but absent from the two other graphs, it is regarded as optional and neutral for detection. Thanks to this complement, we can now, besides detecting all possible forms of instantiation of the pattern, refuse certain forms which we suppose bad for the intent of the fragment.

D. Genericity of the Detection

Our generic detection algorithm is decomposed into three steps (***local_SPC***, closure form the reference participant, ***global_SPC***). At the end of each step, a set of characteristic classes is selected from the set resulting from the preceding step. The first step consists in classifying all classifiers of the model to analyze according to local structural properties of each participant of the pattern. The second step consists in computing potential fragments according to the paths between the reference participant and the others participants of the pattern. The third step consists in verifying the global conformance on the potential fragments and to pinpoint complete fragments. Then the structure of the algorithm is flexible and does not depends on the structure of the pattern to retrieve. The genericity of our approach is provided by an automatic queries generator using profiled patterns. So, to detect a new pattern, it is sufficient to profile it and to generate its query [12].

We chose OCL to encode our detection queries. OCL is a language of constraints used to add semantics into UML models [9]. The Neptune platform was developed by our team within the European Neptune project [10]. The OCL interpreter proposed by the platform implements the standard OCL 2.0 [9] and two extensions of OCL [11]. The second relates to the queries which can return a result of any type of the meta-model, which introduces the concept of view. We use this capacity to carry out our search for complete fragments in a model. Moreover, thanks to the navigational property of OCL, the generation of the queries consists in navigating in the meta-model of the pattern to detect. In following the three steps of the algorithm and in considering the pattern as a graph, the generator analyzes all the possible paths between each vertex and transforms them in an OCL query.

V. VALIDATION

In order to validate the detection algorithm we have sought models of real projects. We would like recall here that we search model fragments at design level without any information from the code. Unfortunately, we did not find industrial projects with exploitable models for our needs; the percentages of association links on inheritance links are too low. So, we have used the code of free projects to obtain reversed models. To do so, we used the Java reverse module of ArgoUML.

The problem with code reversion relates to associations between classes. It is very difficult for reverse softwares to know which variable must be regarded as attribute, association, composition, etc. However, the module of ArgoUML makes it possible to impose that all the attributes are transformed into associations. To take into consideration the parameterized genericity of the latest versions of Java, we have added in the ArgoUML module the capacity to convert these types into (1..n) associations. Thanks to this modification, we consider that the reversed models have a good abstraction level.

In order to make sure of the validity of our detection algorithm, we sought fragments corresponding to characteristic architectures. Fig. 12 presents the fragments that we have searched in nine models (ArgoUML, JUnit, JFreeChart, JabRef, Jena, AWT, JHotDraw, JRefactory, and Neptune). We can notice that these fragments are the *Bridge* pattern and its alternatives. In order to precise the detection, we have searched the same fragments twice: first, with information about authorized and prohibited relations, and second without any complementary information. For these examples, all supplementary relations between classes are prohibited.

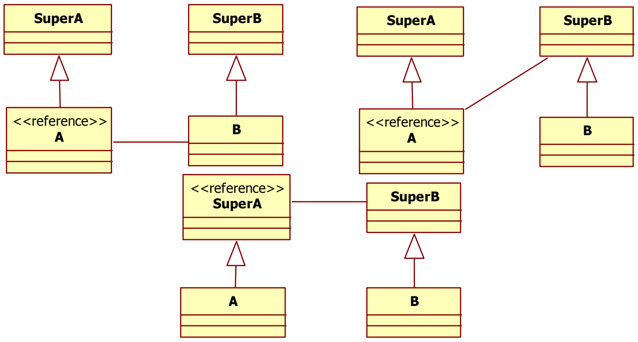


Figure 12. The searched fragments

To validate the queries generated for these micro-architectures, we have executed them directly on the models allowing the generation of the queries. All queries are able to detect the model which has permitted their generation. So, we can say that we detect, at least, the minimal micro-architecture with exact instantiation.

The results obtained at the end of the detection show that all possible instantiations have been identified in the models, and no more. You can see the OCL queries generated corresponding to each pattern of Fig. 12 on [20]. It is interesting to notice that all fragments are identified twice: {*SuperA*, *A*, *B*, *SuperB*}, and {*SuperB*, *B*, *A*, *SuperA*}. The result is correct; each structure pattern is symmetric. Moreover, the researches with information about authorized and prohibited relations have retrieved fewer fragments than with the fragment without complementary information.

VI. Related Works

We present three generic detection techniques used in a similar context. The first two concern approached identifications of design patterns for re-documentation, and the third concerns exact identification of design problems solvable by design patterns.

Patterns identification by comparison of similarities. [1] proposes an algorithm for approximate pattern matching based on comparisons of similarities. This algorithm generates, from two graphs encoded by a matrix, an adjacency matrix representing the closeness of these two graphs. Matrix representations of relationships between artifact developments are used to compute effective approximations. Another way to represent UML models is to consider their visual forms as multi-graphs. Each type of relationship is represented by a different graph. A designer can choose on what information the similarity search is based (associations, generalizations, method specializations, method invocations, etc.). Thus, for a model and a pattern, the average of weighted matrices results for each significant input matrix (associations, generalizations, methods, etc.) is computable. The weighting is determined by the designer according to the importance it wishes to provide the various relationships of a static UML model. This weighted average is then the likeness of a fragment from a design pattern.

Patterns detection by fuzzy evaluation of UML models. Fujaba (From UML to Java And Back Again) is a tool that can generate Java code from a UML model and impact code changes on this model [13] [14]. A component of automatic detection of design patterns has been added. This component uses Abstract Syntax Graphs (ASG) [15] to describe a model by eliminating most syntactic variants and formatting problems. Design patterns are decomposed into sub-patterns implemented as rules of graph transformation. However, if some sub-patterns are generic, eg. all possible ways to assign a value to an attribute in a method, the detection algorithm cannot detect patterns whose shape is not really the sub-assembly patterns provided. To overcome this limit, fuzzy evaluation mechanisms have been proposed by S. Wenzel [2]. They can detect patterns used differently than what is recommended, as well as incomplete patterns. To detect patterns in UML, it is necessary to describe a combination of roles in the UML sense. A role corresponds to a meta-class, which can be attached to OCL constraints [9]. Constraints can describe the complex arrangements of certain patterns and clarify the internal organization of each role. Using this mode of representation, it is possible to detect design patterns in the same manner as a "cast of theater" [2]. The detection assigns a role in the pattern to some model elements. Each assignment is quantified by a value (0 to 100%) representing how the element can play the role. To get 100%, elements must have the same type as the role and respect each of the constraints described in the pattern. After the detection, the candidate fragments are presented to the designer. Implemented in a component of Fujaba, this technique detects target fragments "similar" to contextualization’s design patterns.

Problems detection by constraint propagation. El-Boussaidi and Mili [16] proposes to detect fragments consistent with the meta-model of the problem of a pattern, and replace fragments by instantiation of the corresponding patterns. This detection technique reformulates the problem of homomorphism of graphs proposed by M. Rudolf [17]. A CSP is defined by a finite set of variables in a domain, and a finite set of constraints specifying how values can be assigned to variables [18]. CSPs are generally used to solve efficiently backtracking algorithms. When a value is assigned to a variable, all the constraints of this variable are propagated to other variables. To construct a CSP dedicated to pattern matching, it is necessary to work with two graphs, one to search (the source graph) and one in which research is conducted (the target graph) [17]. Each vertex and each arrow of the source graph are associated with distinct variables. The domain of variable vertices and arrows correspond respectively to the set of vertices and arrows of the target graph. The constraints construction is done on the parameters compared to validate the research. H. Mili and G. El-Boussaidi [19] defines design patterns as a triple (MP, MS, T) where MP is the problem solved by the pattern, MS is the solution to the problem, and T is the transformation that converts MP to MS. MP and MS are respectively the meta-models of the problem and solution. When a designer discovers a fragment of his model conforms to the meta-model of the problem, he has only to apply the transformation rules for modifying the fragment.

TABLE II. SUMMARY OF APPROACHES

	[1]	[14]	[16]	us
Do not perform preprocessing on the model to analyse		OK		OK
Do not use information from the code				OK
Perform detection by successive steps		OK	OK	OK
Limit the execution time			OK	OK
Limit the number of fragments identified			OK	OK
Do not degrade the consensus of the pattern	OK	OK		OK
Detect all possible instantiations	OK	OK		OK

Table 2 summarizes the specifics of our problem with the related works. The last column refers to our approach. We guarantee a technique for early detection, even on large models. It is not pertinent to test all the meta-classes of the model to analyze, and it is better to make a quick filter to restrict the number of comparisons. Then we work directly and without pretreatment with the patterns encoded as models. Moreover, a spoiled pattern constitute a base generating a family of possible instantiations, and we identify accurately all the fragments of the same family.

VII. CONCLUSION

We have presented a complete method to detect generic micro-architectures on models. Then, a major issue of our work is the fact that we have reasoned at design level uniquely. That implies to use information present in models only and to define a re-documentation technique for retrieve patterns in a model. We have used standards dedicated to model engineering: UML profile and OCL queries. From a profiled model representing the structure of a pattern, we deduce automatically the OCL query that permit to retrieve all authorized instantiations of this pattern in a design model.

The detection is the core part of a tooling design review activity [4]. We have implemented this activity into satellite software of the Neptune platform and we named it Triton [12]. As a code review permits to detect inconsistencies, no respect of coding rules, and bad smells in code before a production running, the design review permits to detect model fragments bad conceived and to refactor thanks to design patterns before a coding stage. The results of our queries are not too strict and not fuzzy; each detected fragment can be precisely built from a canonical form and by successive addition of participants. Therefore, we think that our detection method can be reused by the query part of any transformation model language.

However, the detection is based on structural properties only. For now, we have a catalog of spoiled structural design patterns. Dynamic views would be taking into consideration to detect behavioral design patterns and to precise some structural patterns by the detection of message exchange motifs.

REFERENCES

[1] N. Tsantalis and S. T. Halkidis, “Design Pattern Detection Using Similarity Scoring”, *in:* IEEE Transactions on Software Engineering, IEEE Press, volume 32, number 11, pages 896-909, 2006.

[2] S. Wenzel, “Automatic detection of incomplete instances of structural patterns in UML class diagrams”, *in:* Nordic Journal of Computing, Publishing Association Nordic Journal of Computing, volume 12, number 4, pages 379-394, 2005.

[3] H. Albin-Amiot, P. Cointe, Y.-G. Guéhéneuc, and N. Jussien, “Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together”, *in:* proceedings of the 16th conference on Automated Software Engineering (ASE), IEEE Computer Society Press, pages 166-173, 2001.

[4] C. Bouhours, H. Leblanc, and C. Percebois, “Bad smells in design and design patterns”, *in:* Journal of Object Technology, ETH Swiss Federal Institute of Technology, volume 8, number 3, pages 43-63, 2009.

[5] Object Management Group., “Unified Modeling Language”, http://www.omg.org/spec/UML/1.5/PDF/index.htm, 2010.

[6] Object Management Group., “XML Metadata Interchange”, http://www.omg.org/technology/xml/index.htm, 2007.

[7] J. R. Ullmann, “An Algorithm for Subgraph Isomorphism”, *in:* journal of the ACM (JACM), ACM, volume 23, number 1, pages 31-42, 1976.

[8] A. Habel, R. Heckel, and G. Taentzer, “Graph grammars with negative application conditions”, *in:* Fundamenta Informaticae Journal, IOS Press, volume 26, number 3-4, pages 287-313, 1996.

[9] Object Management Group., “Object Constraint Language”, http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf, 2006.

[10] Neptune consortium, “Method, Checking and Document generation for UML applications”, http://neptune.irit.fr/images/files/Neptune Book/407719ps.pdf, 2003.

[11] T. Millan, L. Sabatier, T. T. Le Thi, P. Bazex, and C. Percebois, “An OCL extension for checking and transforming UML Models”, *in:* proceedings of the 8th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS), WSEAS Press, pages 144-150, 2009.

[12] C. Bouhours, “Detection, Explications et Restructuration de défauts de conception : les patrons abîmés”, PhD, IRIT, 2010.

[13] FUJABA, From UML to Java and Back Again, http://wwwcs.uni-paderborn.de/cs/fujaba/projects/ reengineering/index.html, 2005.

[14] S. Wenzel, “Detection of Incomplete Patterns Using FUJABA Principles”, *in:* proceedings of the 3rd International Fujaba Days 2005 : MDD in Practice, pages 33-40, 2005.

[15] J. Niere, W. Schäfer, J.P. Wadsack, L. Wendehals, and J. Welsh, “Towards pattern-based design recovery”, *in:* proceedings of the 24th International Conference on Software Engineering (ICSE), ACM Press, pages 338-348, 2002.

[16] G. El-Boussaidi and H. Mili, “Detecting Patterns of Poor Design Solutions Using Constraint Propagation”, *in:* proceedings of the 11th international conference on Model Driven Engineering Languages and Systems (MoDELS), Springer-Verlag, volume 5301, pages 189-203, 2008.

[17] M. Rudolf, “Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching”, *in:* selected papers from the 6th International Workshop on Theory and Application of Graph Transformations (TAGT), Springer-Verlag, pages 238-251, 2000.

[18] F. Bacchus and P. Van Beek, “On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems”, *in:* proceedings of the 15th National Conference on Artificial Intelligence (AAAI) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI), AAAI Press, pages 311-318, 1998.

[19] H. Mili and G. El-Boussaidi, “Representing and Applying Design Patterns: What Is the Problem?”, *in:* proceedings of the 8th international conference on Model Driven Engineering Languages and Systems (MoDELS), pages 186-200, 2005.

[20] http://www.irit.fr/~Cedric.Bouhours/Examples/

A Benchmark Platform for Design Pattern Detection

Francesca Arcelli Fontana, Marco Zanoni, Andrea Caracciolo
Dipartimento di Informatica, Sistemistica e Comunicazione
University of Milano Bicocca
Milano, Italy
Email: {arcelli,marco.zanoni}@disco.unimib.it, a.caracciolo1@campus.unimib.it

***Abstract*—Design patterns detection is a useful activity in reverse engineering to gain knowledge on the design issues of an existing system, on its software architecture and design quality, improving in this way the comprehension of the system and hence its maintainability and evolution. Several tools have been developed, but they usually provide different results analyzing the same systems. Some works have been proposed in the literature to compare these results, but a standard widely accepted benchmark is not yet available. In this work we propose our benchmark platform for design patterns detection, based on a community driven evaluation.**

***Keywords*-design pattern detection; benchmark**

I. Introduction

Design pattern detection (DPD) is a topic which received a great interest during the last years. Finding design patterns (DPs) [1] in a software system can give very useful hints on the comprehension of a software system and on what kind of problems have been addressed during the development of the system itself. Moreover, they are very important during the re-documentation process, in particular when the documentation is very poor, incomplete or not up-to-date.

Several DPD approaches and tools have been developed both for forward and reverse engineering aims and involving different techniques for the detection such as fuzzy logic, constraints solving techniques, theorem provers, template matching methods and classification techniques (i.e., [2], [3], [4], [5], [6], [7], [8], [9], [10]). In spite of the many approaches proposed, the results obtained are often quite unsatisfactory and different from one tool to the other.

Many tools find many false positive instances but other correct instances are not found. One common problem in DPD is the so called variant problem: DPs can be implemented in several ways, often very different from one another. The main variants for each pattern are described in the catalog of [1], others are applied when the context of application requires it. These variations cause the failure of most pattern instances recognition using rigid detection approaches, which are based only on canonical pattern instances.

Hence the comparison of the results provided by the different tools is very important, in order to be able to evaluate the best approach and technique. Some works that have been proposed respect to the comparison of DPD tools are described in the next section.

We analyzed and faced the problem related to the different results provided by the DPD tools, since we are developing a tool called MARPLE (Metrics and Architecture Reconstruction plug-in for Eclipse) [11] whose main aims are related to DPD and software architecture reconstruction. The Marple DPD module is characterized by the following steps:

- the detection of sub components or micro structures, which give useful hints on the DP detection, with the aim of mitigating the variant problem;
- the detection of the largest possible set of DP candidates performed by a module called Joiner, whose results are characterized by very high recall values;
- the refining of the previous results through data mining techniques, in particular through a step of clustering and a step of supervised classification.

The aim of this work is to introduce and describe a benchmark platform to be used to compare DPD tools. We propose some mechanisms to obtain safer results and to make them available to the DPD community in an easy way. Our approach is characterized by:

- a general model for design pattern representation;
- a way of comparing results coming from different tools;
- the possibility to evaluate instances and discuss about their correctness.

The adoption by the DPD community of a benchmark can improve the cooperation among the researchers and the reuse of tools written by other instead of the development of new ones.

II. Related Work

As we observed before, undertaken a comparison among design pattern detection tools is certainly a difficult task. Few benchmark proposals for the evaluation of design pattern detection tools have been presented in the literature. In [12], we describe our first proposal for a benchmark, that we extend and finalize in this paper. In [13], the authors present their work in progress towards creating a benchmark, called DEEBEE (DEsign pattern Evaluation Benchmark Environment), for evaluating and comparing design pattern detection tools. Currently, the benchmark database contains the results of three DP tools.

In [14], it has been defined P-MARt, a repository of pattern-like micro-architectures, with the purpose to serve as baseline to assess the precision and recall of pattern identification tools. The repository contains the analysis of specific version of some different open source projects. While in [15] the authors compare different design patterns detection tools and they propose a novel approach based on data fusion, build on the synergy of proven techniques, without requiring any re-implementation of what is already available. In [16], the authors develop DPDX, a rich common exchange format for DPD tools, to overcome limitations due to the different output format. DPDX provides the basis for an open federation of tools that perform comparison, fusion, visualization, and-or validation of DPD results.

In our work we aim to provide an online tool to support the comparison of DPD results, with the benefit of having a flexible underlying model and a collaborative environment. In future work we will experiment the correctness and completeness of our approach exchanging data with the above cited platforms and models.

III. DP REPRESENTATION

In order to work on design pattern instances we need a way to represent them in some kind of data structure. In [12], we presented a model for the representation of DP definitions and instances. That model now stands behind all the elaboration made by the benchmark platform.

The model specifies that a design pattern can be defined from the structural point of view using the roles it contains and the cardinality relationship between couple of roles. A design pattern is defined as a tree whose nodes are called *Levels*; each Level has to contain at least one of the roles of the pattern and it can contain other nested Levels, recursively. In Figure 1 it is possible to see the tree structure of the *LevelDef* class (representing the level definition), and the *RoleDef*s it owns; finally, *DpDef* defines that a design pattern definition is a tree having as root one *LevelDef*.

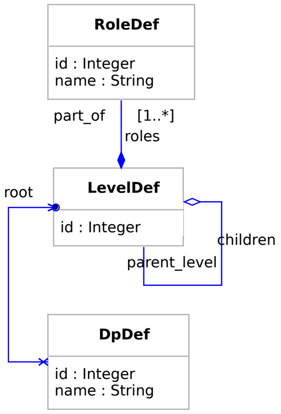


Figure 1. DP Definition UML class diagram

When two roles are contained in the same level, they are in a one-to-one relationship; instead when a role is in a nested Level it means that for each instance of the roles in the parent level, there can be many sets of roles of the child level. The most common case is when a pattern defines that a class must extend another class. In most cases we identify a single instance of that pattern as the parent class connected with all the children classes. Instances are modeled as in Figure 2; the model is simply an extension of the definition, as it models the instantiation of the concepts contained in the definition: a *RoleAssociation* is the realization of a *RoleDef*, a *LevelInstance* is the realization of a *LevelDef*, and so on. The only complex detail is the splitting of *Level* and *LevelInstance*; the explanation is that each *LevelDef* is instantiated as a *LevelInstance* when the *RoleAssociation*s are filled, but to define a child *Level* we need to specify which particular parent instance it belongs to.

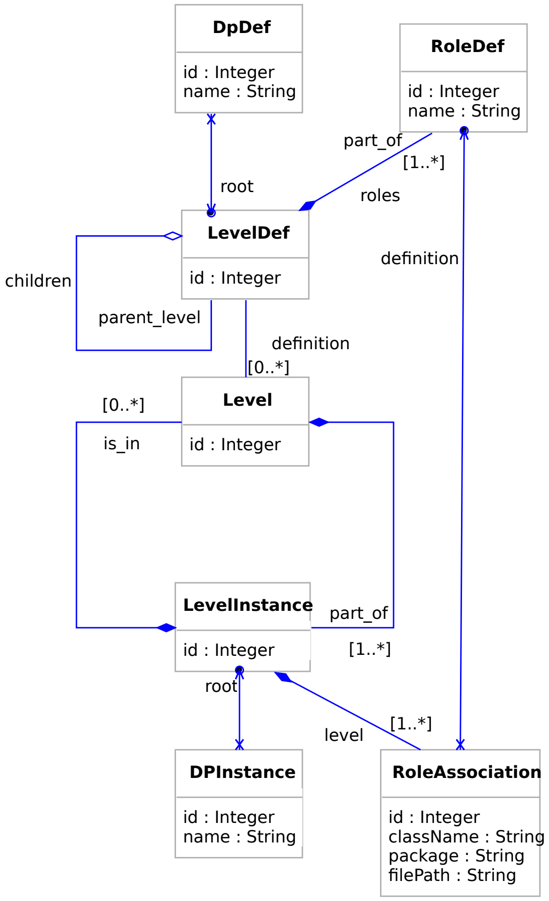


Figure 2. Model UML class diagram

A. XML format

The XML format for specifying design pattern instance is modeled according to the representation model, so it follows the same concepts. The XML Schema Definition, which the submitted XML file must comply to, is available at [17]; moreover, the list of the pattern definitions the tool can support is also available at [18].

In order to have more chances of being able to compare results coming from different tools, currently users cannot supply their own pattern definitions. When the platform will be more tested and filled with data, we will add this functionality. Meanwhile we accept and encourage suggestions coming from the users about new definitions or mistakes in the current ones.

In the following we report an XML file example that represents an instance of an Abstract Factory design pattern.

Listing 1. Example of Abstract Factory instance

```
<analysis
xmlns=”http://www.essere.disco.unimib.it/DPBWeb”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema–
    instance”
xsi:schemaLocation=”http://www.essere.disco.unimib
    .it:8080/DPBWeb/resources/DpAnalysis.xsd”>
<pattern name=”Abstract␣Factory”>
<patternInstance>
  <role name=”Abstract␣Factory”
     package=”org.foo.bar”
     class=”AbstractFactoryClassName”
     filePath=”org/foo/bar/a1.java” />
  <level>
    <levelInstance>
      <role name=”Concrete␣Factory”
         package=”org.foo.bar”
         class=”ConcreteFactoryClassName”
         filePath=”org/foo/bar/b1.java” />
    </levelInstance>
  </level>
  <level>
    <levelInstance>
      <role name=”Abstract␣Product”
         package=”org.foo.bar.baz”
         class=”AbstractProductClassName”
         filePath=”org/foo/bar/baz/c1.java” />
      <level>
        <levelInstance>
          <role name=”Concrete␣Product”
             package=”org.foo.bar.baz”
             class=”ConcreteProductClassName1”
             filePath=”org/foo/bar/baz/d1.java” />
        </levelInstance>
        <levelInstance>
          <role name=”Concrete␣Product”
             package=”org.foo.bar.baz”
             class=”ConcreteProductClassName2”
             filePath=”org/foo/bar/baz/d2.java” />
        </levelInstance>
      </level>
    </levelInstance>
  </level>
  <level>
    <levelInstance>
      <role name=”Client”
         package=”org/foo/bar/baz”
         class=”ClientClassName”
         filePath=”org/foo/bar/baz/e1.java” />
    </levelInstance>
  </level>
</patternInstance>
</pattern>
</analysis>
```

The file refers to the roles *Abstract Factory*, *Concrete Factory*, *Abstract Product*, *Concrete Product*, each associated to a class name and to a package name, and organized following the Abstract Factory definition.

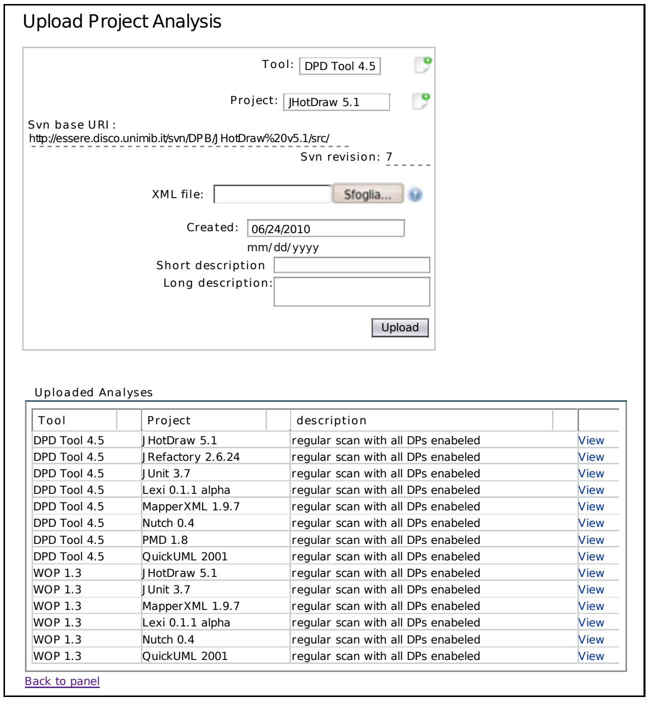
IV. DPD PLATFORM

The platform is available at [19] and it is subdivided in:

- the *documentation* section contains some references and guides to use the platform; in addiction the home page briefly introduces the system functionalities and provides a step by step tutorial;
- the *search* section lets the user to find the results of a particular analysis, according to different parameters;
- the *compare* section allows to compare the instances found by different tools on the same input project;
- the *browse* section provides a tree-like view of the contents of the platform.

Through these different kinds of exploration the user can obtain the detailed view of each pattern instance, that includes two different types of graphic visualization and a simple forum for the *evaluation* of the instance by the users.

All the above functionalities are visible to all users; if a user wants to load new pattern instances into the platform he must be registered, log into the platform, and submit the XML file containing the instances and some metadata, as shown in Figure 3. In our platform an *analysis* consists of the combination of the set of the instances, its description, the choice of the DPD tool and the analyzed project.



Upload Project Analysis

Tool: DPD Tool 4.5

Project: JHotDraw 5.1

Svn base URI :
http://essere.disco.unimib.it/svn/DPB/JHotDraw%20v5.1/src/

Svn revision: 7

XML file: Sfoglia...

Created: 06/24/2010
mm/dd/yyyy

Short description

Long description:

Upload

Uploaded Analyses

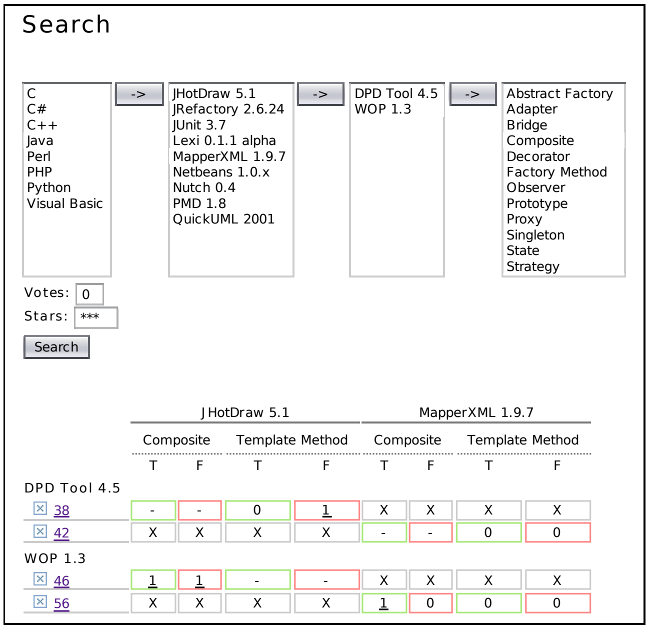
Tool	Project	description	
DPD Tool 4.5	JHotDraw 5.1	regular scan with all DPs enabeled	View
DPD Tool 4.5	JRefactory 2.6.24	regular scan with all DPs enabeled	View
DPD Tool 4.5	JUnit 3.7	regular scan with all DPs enabeled	View
DPD Tool 4.5	Lexi 0.1.1 alpha	regular scan with all DPs enabeled	View
DPD Tool 4.5	MapperXML 1.9.7	regular scan with all DPs enabeled	View
DPD Tool 4.5	Nutch 0.4	regular scan with all DPs enabeled	View
DPD Tool 4.5	PMD 1.8	regular scan with all DPs enabeled	View
DPD Tool 4.5	QuickUML 2001	regular scan with all DPs enabeled	View
WOP 1.3	JHotDraw 5.1	regular scan with all DPs enabeled	View
WOP 1.3	JUnit 3.7	regular scan with all DPs enabeled	View
WOP 1.3	MapperXML 1.9.7	regular scan with all DPs enabeled	View
WOP 1.3	Lexi 0.1.1 alpha	regular scan with all DPs enabeled	View
WOP 1.3	Nutch 0.4	regular scan with all DPs enabeled	View
WOP 1.3	QuickUML 2001	regular scan with all DPs enabeled	View

Back to panel

Figure 3. Example of analysis loading

A. *Search*

The user can search a pattern instance according to the chosen programming language, project, detection tool, and design pattern. An example of a search section is shown in Figure 4. In the user interface, whenever the user chooses a filter, the page shows the available further filters. In each selection list it is possible to choose more than one value, to make the search more flexible and personalized.



Search

C
C#
C++
Java
Perl
PHP
Python
Visual Basic

JHotDraw 5.1
JRefactory 2.6.24
JUnit 3.7
Lexi 0.1.1 alpha
MapperXML 1.9.7
Netbeans 1.0.x
Nutch 0.4
PMD 1.8
QuickUML 2001

DPD Tool 4.5
WOP 1.3

Abstract Factory
Adapter
Bridge
Composite
Decorator
Factory Method
Observer
Prototype
Proxy
Singleton
State
Strategy

Votes: 0
Stars: ***

	JHotDraw 5.1				MapperXML 1.9.7			
	Composite		Template Method		Composite		Template Method	
	T	F	T	F	T	F	T	F
DPD Tool 4.5								
☒ 38	-	-	0	1	X	X	X	X
☒ 42	X	X	X	X	-	-	0	0
WOP 1.3								
☒ 46	1	1	-	-	X	X	X	X
☒ 56	X	X	X	X	1	0	0	0

Figure 4. Example of a pattern instance search result

Figure 4 shows an example of search parameters and results, in the case the user looks for the analysis of project written in the Java language, on the "JHotDraw 5.1" and "MapperXML 1.9.7" projects, analyzed by the "DPDTool 4.5" and "WOP 1.3" tools and filtered on the "Composite" and "Template Method" design patterns.

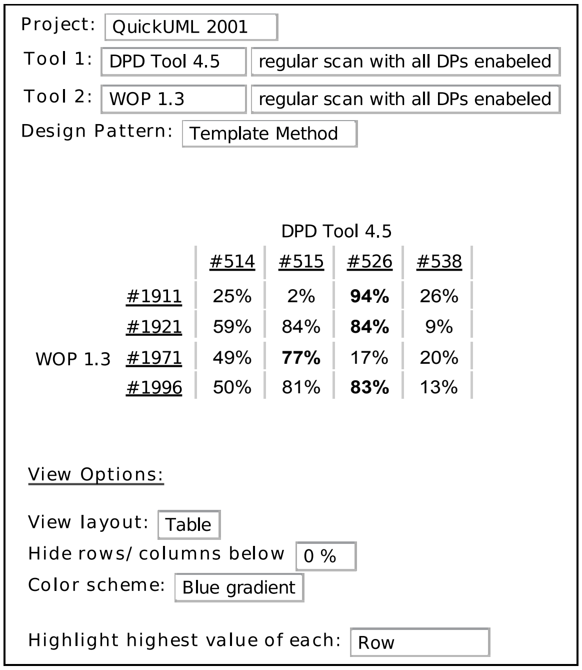
The table (see Figure 4) shows the results with the tools and their analysis on the rows and the projects with the patterns on the columns. Each cell is the combination of an analysis of a tool and a pattern belonging to a project, and it contains the number of instances considered correct and the number of the ones considered incorrect, respectively under the column label "T" and "F".

The bias value that allows the platform to choose if an instance is correct or not is specified by the two combo boxes named "Votes" and "Stars". The details of this topic are explained in Section IV-D.

There is a special case regarding the cell values: for example, when an analysis does not contain instances of a particular design pattern in a particular project, the corresponding cells are rendered as "X" in light grey.

B. *Comparison*

The user can compare the results produced by two different analysis, obtained by two different tools, on the same project and for the same single chosen pattern definition. The comparison results (see Figure 5) are shown in a table where the two instance sets, found by the two analysis, are shown one on the rows and the other on the columns. The cells of the table contain values indicating the similarity of the corresponding couple of instances. The similarity is currently evaluated through a very simple algorithm described below. The background color of each cell is proportional to its percentage value, according to the selected color scheme (red:0% to green:100% or white:0% to blue:100%).



Project: QuickUML 2001
Tool 1: DPD Tool 4.5 | regular scan with all DPs enabeled
Tool 2: WOP 1.3 | regular scan with all DPs enabeled
Design Pattern: Template Method

		DPD Tool 4.5			
		#514	#515	#526	#538
WOP 1.3	#1911	25%	2%	**94%**	26%
	#1921	59%	84%	**84%**	9%
	#1971	49%	**77%**	17%	20%
	#1996	50%	81%	**83%**	13%

View Options:

View layout: Table
Hide rows/columns below 0 %
Color scheme: Blue gradient
Highlight highest value of each: Row

Figure 5. Example of a result comparison

Figure 5 shows an example of comparison: the user selects to compare the instances of the Composite pattern found on project "QuickUML 2001" by:

- the "DPD Tool 4.5" tool in the analysis named "regular scan with all DPs enabled";
- the "WOP 1.3" tool in the analysis named "regular scan with all DPs enabled".

The table shows that each tool found four instances, and that for example the instances #1911 and #526 are very similar, with a score of 94%, and instances #1911 and #515 are very different, having a score of only 2%. The numbers in bold are the highest value of the rows: in fact the last option selected is to highlight the highest value of each row; it is also possible to do the same on the columns. This option simplifies the task of finding the instances that are more similar in order to understand if they are really the same.

The view can be also filtered removing all the rows or columns having all the values less or equal to the one specified, in order to simplify the table and to include only meaningful results.

1) Comparison algorithm: In the current version of the benchmark platform we implemented a comparison algorithm we developed as a proof of concept. The algorithm tries to express the similarity of two instances giving more weight to the parent roles and less to the children roles, and produces a number between 0 (total difference) and 1 (total equivalence). The actual version of the algorithm is very simple and any suggestions and contribution coming from the DPD community, in order to populate the platform with other comparison algorithms, is welcome.

Our algorithm gives a descending score to each level depth in the pattern definition: for example in a definition with only a parent level and a child level, the parent level depth takes a score of 2 and the child level depth takes a score of 1; with three level depths the first one (the root) would take weight 3, and so on. Then each level in the definition takes the score of the depth it belongs to, and an overall score is calculated as the sum of the score of all levels.

Finally a weight is assigned to each level dividing the level score by the overall score. In this way the sum of all the weights is 1.

When the weights are set, the algorithm compares a couple of instances starting from the root and recursively distributing the weight of each level on its level instances; we consider that two level instances are equal if their sets of role association are equal.

For example: let us say a level weights $1/2$ and the two levels to compare contain the first one instance and the second two instances; in addition the instance in the first level is equal to one of the two in the second one level. Their direct comparison value is $1/2$, because we have only one match out of two, that is the maximum of the number of the instances in each level. Then the overall score of the comparison is $1/2*1/2$, because we have to weight the local score with the global weight.

The overall score of the comparison is the sum of all the local scores.

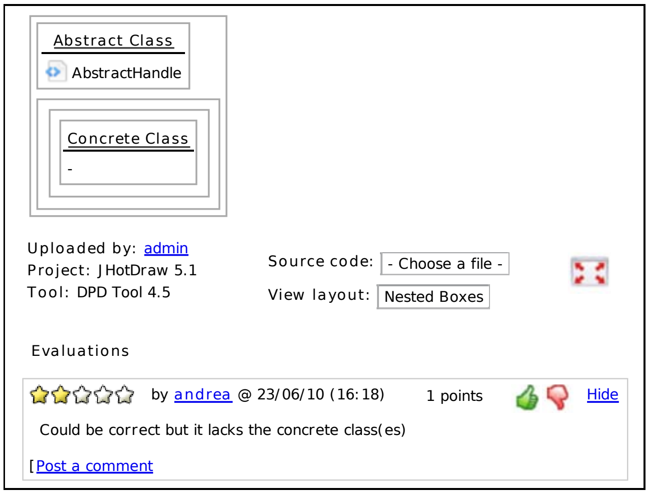
C. Browse

The Browse section offers a tree view of the content of the platform. Basically it allows the user to find all the analysis made by a tool on a project: the user can choose as entry point both the available projects or tools. Whenever the users clicks on one of them, the page shows the analysis available in combination with the other.

For example, clicking on the name of the analysis, the platform shows the analysis details page containing the list of the found DP instances, grouped by design pattern.

D. Evaluation

The evaluation phase allows the user to analyze the reported instance. The instance can be graphically viewed in two ways: in the first way using a graph representation of the tree representing the instance (it requires java working in the browser), and in the second way using nested boxes (see Figure 6) to represent the nested structure of the tree representing the pattern instance.



Abstract Class
AbstractHandle

Concrete Class
-

Uploaded by: admin
Project: JHotDraw 5.1
Tool: DPD Tool 4.5

Source code: - Choose a file -
View layout: Nested Boxes

Evaluations

★★☆☆☆ by andrea @ 23/06/10 (16:18) 1 points Hide

Could be correct but it lacks the concrete class(es)

[Post a comment

Figure 6. Example of the visualization of an instance

The rest of the evaluation phase is concerned with the discussion forum about the found instance. This forum is dedicated to the evaluation of the correctness of the found instance:

- it allows to express a score (the “Stars”) in the range 1-5, where 1 means the instance is fully incorrect, and 5 means it is correct; it is also possible to insert a comment to argument the evaluation;
- it lets other users to express an agreement or disagreement (the “Votes”) with previous evaluations.

The overall scoring is used as a parameter in the search page (see Figure 4) to have an immediate idea about the overall correctness of a found instance.

In fact, during the search phase, the user can specify the minimum number of stars a pattern instance must have in order to be considered a correct instance. In addition, the user can specify the absolute number of agreements (or disagreements) each evaluation must have to be safely included in the overall stars computation. The overall stars number of an instance is computed as the average star number weighted with the agreement balance (the difference between the number of agreements and disagreements), considering only the evaluations having the minimum number of absolute agreements. Currently, the platform allows the users to choose the number of stars in the range 1-5, and the number of agreements (called “Votes” in the platform) from 0 to 20.

We believe that this community driven system of classifying found instances provides good common datasets, for the test of new tools and the enhancement of the existing ones.

Moreover, in the evaluation phase it is possible to inspect directly the source code, selecting the file to inspect from the combo box under the graphical representation of the pattern.

V. CONCLUSION

In this paper we presented a platform helping the design pattern detection community having a way to compare the results produced by the tools and the techniques that have been proposed in the literature.

Our final intent is not only the tool "competition" but also the creation of a container for design pattern instances that, through the users' voting, will allow us to build a large and "community validated" dataset for tool testing and benchmarking.

For all these reasons we are convinced that this kind of platform can be really valuable in our research area because it allows the real sharing of information and knowledge among all research groups interested in design patterns for both reverse and forward engineering.

In future work we are interested in integrating different comparison algorithms, maybe suggested and discussed with the DPD community, in order to let the users to choose the algorithm they think is the more appropriate; in addition, we need to refine and tune the platform settings. We are also investigating the possibility to expose some kind of web services in order to let registered users to make their tool able to automate the loading of their analysis into the platform.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[2] A. D. Lucia, V. Deufemia, C. Gravino, and M. Risi, "Design pattern recovery through visual language parsing and source code analysis," *Journal of Systems and Software*, vol. 82, no. 7, pp. 1177 – 1193, 2009.

[3] H. Huang, S. Zhang, J. Cao, and Y. Duan, "A practical pattern recovery approach based on both structural and behavioral analysis," *Journal of Systems and Software*, vol. 75, no. 1-2, pp. 69 – 87, 2005, software Engineering Education and Training.

[4] M. von Detten, M. Meyer, and D. Travkin, "Reverse engineering with the reclipse tool suite," in *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. New York, NY, USA: ACM, 2010, pp. 299–300.

[5] R. A. Olsson and N. Shi, "Reverse engineering of design patterns from java source code," in *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 123–134.

[6] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 896–909, 2006.

[7] Y.-G. Guéhéneuc, "Ptidej: Promoting patterns with patterns," in *Proceedings of the 1st ECOOP workshop on Building a System using Patterns*, M. E. Fayad, Ed. Springer Verlag, July 2005, 9 pages.

[8] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery," in *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002, pp. 338–348.

[9] J. Dietrich and C. Elgar, "Towards a web of patterns," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 108 – 116, 2007, software Engineering and the Semantic Web.

[10] Y.-G. Gueheneuc and G. Antoniol, "DeMIMA: A multi-layered approach for design pattern identification," *IEEE Transactions on Software Engineering*, vol. 34, pp. 667–684, 2008.

[11] F. Arcelli, C. Tosi, M. Zanoni, and S. Maggioni, "The marple project - a tool for design pattern detection and software architecture reconstruction," in *Proceedings of the International Workshop on Advanced Software Development Tools and Techniques (WASDeTT 2008)*, Paphos, Cyprus, July 2008.

[12] F. Arcelli, C. Tosi, and M. Zanoni, "A benchmark proposal for design pattern detection," in *FAMOOSr 2008: Proceedings of 2nd Workshop on FAMIX and Moose in Reengineering*, 2008.

[13] L. Fulop, R. Ferenc, and T. Gyimothy, "Towards a benchmark for evaluating design pattern miner tools," in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, 1-4 2008, pp. 143 –152.

[14] Y.-G. Guhneuc, "Pmart: Pattern-like micro architecture repository," in *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*, M. Weiss, A. Birukou, and P. Giorgini, Eds., July 2007.

[15] G. Kniesel and A. Binun, "Standing on the shoulders of giants - a data fusion approach to design pattern detection," in *ICPC*. IEEE Computer Society, 2009, pp. 208–217.

[16] G. Kniesel, A. Binun, P. Hegedűs, L. J. Fülöp, N. Tsantalis, A. Chatzigeorgiou, and Y.-G. Guéhéneuc, "A common exchange format for design pattern detection tools," in *CSMR 2010*, March 2010.

[17] ESSeRE, "Design pattern analysis schema definition," Web Site, 2010, http://essere.disco.unimib.it:8080/DPBWeb/faces/resources/DpAnalysis.xsd.

[18] ——, "Design pattern definitions documentation," Web Site, 2010, http://essere.disco.unimib.it:8080/DPBWeb/faces/Doc_DpDef.jsp.

[19] ——, "Design pattern benchmark platform," Web Site, 2010, http://essere.disco.unimib.it/DPB/.

Tissue Classification from CT of Liver Volumetric Dataset Using 3D Relational Features

Wan Nural Jawahir Hj Wan Yussof
Chair for Pattern Recognition and Image Processing
Albert-Ludwigs-Universität Freiburg
Freiburg im Breisgau, Germany
yussof@informatik.uni-freiburg.de

Hans Burkhardt
Chair for Pattern Recognition and Image Processing
Albert-Ludwigs-Universität Freiburg
Freiburg im Breisgau, Germany
Hans.Burkhardt@informatik.uni-freiburg.de

***Abstract*—This paper proposes an extension of two dimensional relational features into three dimensions. In two dimensions, the relational features are extracted using a non-linear kernel function. This function is applied to the values of the points of two circles. To extract 3D relational features, we represent the points on the two spheres. We aim at classifying the tissue from Computed Tomography (CT) of liver datasets into three classes; normal, abnormal and others(i.e., kidneys, blood vessel, etc.). For this task, 100 known points from 6 CT datasets were used for training using the Support Vector Machine (SVM) classifier and 150 points from 10 CT datasets were used for validation. The results presented in this paper show that the relational features are promising.**

***Keywords*-invariant features, relational kernels, Computed Tomography (CT);**

I. Introduction

Image data contains meaningful information that has to be automatically extracted using computers or electronic devices. Such image information are called image features. Depending upon the particular task, the extracted features capture morphological properties, color properties, or certain textural properties of the image.

Based on the collection of texture definitions in [6], formulated by different vision researchers, it is difficult to express the true meaning of texture. However, the utilization of texture features is of increasing interest in many domains, including in the medical domain. Major categories of texture can be grouped into three subfields: texture segmentation, texture classification and texture synthesis. Texture segmentation works on partitioning the differently textured regions in an image. Taking advantage of structural content from a small digital sample image, one can construct a large image. This process is called texture synthesis. In texture classification, the goal is to assign an unknown sample image to one of the known texture classes. We attract the attention of the reader to [6], as these categories have already been discussed in detail by the authors.

This paper deals with the classification problem and presents a method for distinguishing normal tissue from abnormal tissue based on three dimensional textures using CT data recordings of liver datasets. The organization of this paper is as follows: In Section 2, we will review the role of texture features used in medical image processing. In Section 3, we describe the proposed texture feature. Section 4 presents and discusses the results and finally we give the conclusion in Section 5.

II. Texture Features in Medical Applications

Texture features have been applied in many medical image analysis and computer vision problems [1][2][11]. In general, the applications involve the automatic feature extraction from the image. The features are then used for a variety of medical tasks such as classification, segmentation, registration and medical images indexing and retrieval.

Kovalev et al. [11] proposed an extended co-occurrence descriptor for three dimensional texture analysis of MRI datasets. The method is based on extended multi-sort co-occurrence matrices that combine intensity, gradient and anisotropy image features. They have demonstrated that their method is an efficient tool in various MRI image analysis tasks such as classification of brain datasets and segmentation of diffuse brain lesions.

For the registration task, Jarc et al. [1] extracted Laws texture coefficients and used them for computing registration criterion functions. The purpose of their work is to analyze the importance of texture information for registration of a digitaly reconstructed radiograph (DRR) and medical electronic portal image (EPI). They computed a registration criterion function directly from the intensity values, i.e., gray-values, for comparison to their proposed feature based approach. Three observations have been done; the accuracy of registration, the distinctiveness of local extrema and the distinctiveness of a global extrema of the criterion functions. These parameters are essential to achieve a correct image alignment. From the given image modalities, a more robust and correct registration can be expected using texture based instead of using intensity based criterion functions.

Manduca et al. [2] used texture features for the prediction of breast cancer from mammographic images. They extracted five textural features. For each image, a total of 1,050 Markovian texture features, 112 run length features, 250 Laws features and 30 Wavelet features were extracted from the central area of the breast where the thickness is nearly uniform and is referred as constant thickness region (CTR). A total of 41 Fourier features were calculated within the CTR-box. The CTR-box was obtained from the largest rectangular box that can be inscribed within the breast region. For the evaluation they used 381 datasets for training and 387 datasets for validation. The results consistently show that texture features at low spatial frequencies provided the strongest predictors of future breast cancer risk.

Tesar et al. [4] proposed a texture-based segmentation of organs for disease diagnostic. They proposed the extension of 2D Haralick texture features to 3D. For this work, they calculated a separate co-occurrence matrix for each voxel in the 3D image. The co-occurrence matrix is calculated from all voxels in a small rectangular window around the voxel. This makes it possible to segment given a 3D image as opposed to calculating the feature for the pre-segmented regions of an image. Consequently, such features can be used to search for very small regions with different texture properties (like tumors). A set of abdomen CT images was used for evaluation of the proposed approach. They used a Gaussian Mixture Model for the segmentation and for learning the parameters of the mixture models from the training dataset, an expectation-maximization (EM) approach was used.

In medical retrieval applications, Glatard et al. [9] used a bank of Gabor filters, one of the most popular texture features for medical image indexing and retrieval with a database of a cardiac magnetic resonance images. Each filter was tuned to a specific orientation and spatial frequency. Gabor filters with an angular spacing of 30° (corresponding to the orientations $0°, 30°, 60°, 90°, 120°$ and $150°$) and a frequency spacing of one octave (corresponding to the frequencies $\sqrt{2}, 2\sqrt{2}, ..., \frac{N}{4}\sqrt{2}$ cycles per image, N being the size of the image) were calculated. 42 Gabor filters were used for image indexing and 16 Gabor filters were used for segmentation assisted retrieval.

We discovered from the literature review that texture features are an important property in many medical applications. However, among texture features, relational features have not received much attention in this field. In this paper, our intention is to expose the potential of the proposed texture features to the researchers who are enthusiastic in medical image analysis.

III. Relational Invariant Features

In this work, we focus on the classification of liver tissue from datasets based on extended relational features. These features have been successfully used in the SIMBA (Search IMages By Appearance) system[1]. The advantage of using this method is that it is insensitive to image noise and also invariant to a group of image transformations (e.g.,translation and rotation). Using a slice-by-slice two-dimensional approach is still possible but it suffers from the drawback of some important information loss. To benefit from all information in 3D space, we extend the relational features to three dimensions. Note that, since CT data normally comes with low resolution and high noise, our method does not directly work on a raw volume data. We filtered the data using 3D anisotropic diffusion (see Figure 1).

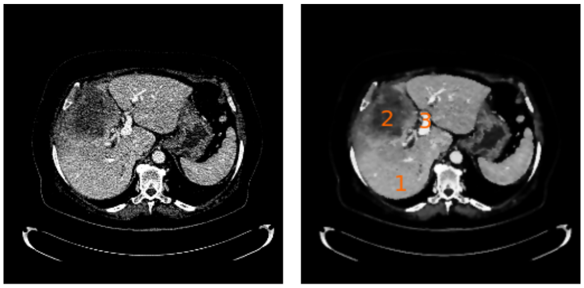


Figure 1. CT scans showing the low-density metastases in the upper right lobe of the liver. These tissues are labeled as 2 (abnormal). The blood vessels which are labeled as 3 (others) also appear as if they have been enhanced with contrast media and are brighter than 1 (normal) liver tissues. Image on the right is the result after noise filtering using anisotropic diffusion from raw data on the left.

A. Invariant Features

In two dimensions, a gray value image, X are represented as $X(x, y), (0 \leq x < M, 0 \leq y < N)$ with $X(x, y)$ be the gray-value at pixel coordinate (x, y) and MxN is the image domain. Let $\mathcal{G}$ be the transformation group of translation and rotation with elements $g \in \mathcal{G}$ acting on the images. The transformed images are gX. An invariant feature must satisfy $F(gX) = F(X), \forall g \in \mathcal{G}$. Integrating $f(gX)$ over the transformation group $\mathcal{G}$ constructs such invariant features as follows:

$$I(X) = \frac{1}{|\mathcal{G}|} \int_{\mathcal{G}} f(gX) dg \tag{1}$$

Eq. 1 becomes

$$\mathrm{IF}(X) = \frac{1}{2\pi MN} \int_{x=0}^{M} \int_{y=0}^{N} \int_{\theta=0}^{2\pi} f(g(x, y, \theta)X) d\theta dx dy \tag{2}$$

when applying the integration over all possible rotations and translations (Haar integral over the Euclidean motion). In discrete form the Eq. 2 is defined as:

$$\mathrm{IF}(X) \approx \frac{1}{qMN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \sum_{j=0}^{q-1} f(g(x, y, \theta = j\frac{2\pi}{q})X) \tag{3}$$

where IF is approximated by choosing x and y to be integers and by varying θ in a discrete manner producing q samples. Interpolation is used to solve the problem of points that do not lie on the image grid.

[1] http://simba.informatik.uni-freiburg.de/

As can be observed, Eq. 3 can be computed locally by applying the kernel function f, on the neighborhood of each pixel in the image for all possible values of θ.

$$\mathrm{IF}_{\mathrm{local}}(x,y) = \frac{1}{q}\sum_{j=0}^{q-1}(f(g(x,y,\theta=j\frac{2\pi}{q})X)). \qquad (4)$$

The summation over the angle θ can be replaced with histogramming [8].

$$\mathrm{IF}_{\mathrm{local}}(x,y) = \mathrm{hist}(f(g(x,y,\theta=j\frac{2\pi}{q})X)), j=0,...,q-1. \qquad (5)$$

In this way, local information can be preserved and thus increasing the discrimination capabilities of features.

B. Relational Features

The relational features are calculated similar to Local Binary Pattern (LBP) texture features [10]. LBP thresholds the neighborhood with the gray value of its center pixel and represents the result as a binary pattern (0 and 1). Applying this to all pixels in a circular neighborhood of the center pixel, the binary pattern is then transformed into a unique number as follows:

$$\mathrm{LBP} = \sum_{i=0}^{n-1} s(v_i - v_c)2^i, \text{where} \qquad (6)$$

$$s(x) = \begin{cases} 1 & x \geq 0, \\ 0 & x < 0 \end{cases} \qquad (7)$$

where v_i and v_c are the gray values at a neighboring pixel and at the center pixel, respectively. The number of the pixels in the circular neighborhood is denoted by n. The drawback of LBP is that the discontinuity of the LBP operator (the s function), since it maps to 0 or 1 which makes them sensitive to noise. A small disturbance in the image may cause a big deviation of the feature.

Using the following form

$$f(X) = \mathrm{rel}(X(x_1,y_1) - X(x_2,y_2)) \qquad (8)$$

we construct an invariant feature by applying Eq. 8 onto Eq. 4. Schael [5] has introduced a ramp function that extends the step function in Eq. 7 giving values in the range of $[0,1]$ as follows:

$$\mathrm{rel}(\eta) = \begin{cases} 1 & \text{if } \eta < -\varepsilon \\ \frac{\varepsilon-\eta}{2\varepsilon} & \text{if } -\varepsilon \leq \eta \leq \varepsilon \\ 0 & \text{if } \varepsilon < \eta \end{cases} \qquad (9)$$

where ε is a threshold parameter. Using a ramp function, it is now more robust to image noise. Note that, if ε is set to zero, then the rel function will reduce to the simple LBP operator s.

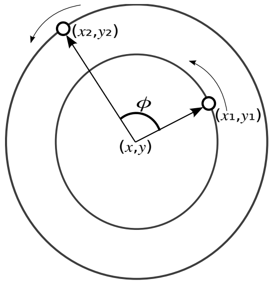


Figure 2. Calculation of a set of relational features in two dimensions. A feature is formed by applying the relational function to the gray-value difference of the pixels lying on the specific distance and phase to the reference point (i.e. center of the circles)

As opposed to LBP, 2D relational features use two circular sets. Let (x,y) be the coordinates of central pixel in two-dimension, taking into consideration a phase shift, ϕ, the coordinates (x_1,y_1) and (x_2,y_2) in Eq. 8 are given by:

$$(x_1,y_1) = (x + r_1\cos(\theta), y + r_1\sin(\theta)) \qquad (10)$$

$$(x_2,y_2) = (x + r_2\cos(\theta+\phi), y + r_2\sin(\theta+\phi)) \qquad (11)$$

where r_1 and r_2 are the radii of the first and second circle, respectively. Local information at different scales and orientations can be captured with different combinations of r_1, r_2 and ϕ. The calculation of two dimensional relational features is illustrated in Figure 2

C. Extension Relational Features to Three Dimension

In this paper, we present an extension of relational features to three dimensions. The relational function in Eq. 8 for three dimensions is very straightforward and is given by:

$$f(X) = \mathrm{rel}(X(x_1,y_1,z_1) - X(x_2,y_2,z_2)) \qquad (12)$$

We have seen that LBP and 2D relational features use circular sets to represent the neighborhood of a central pixel. To extent relational features to three dimensions, a logical way is to represent neighbors in unit sphere. Thus for a central voxel with the coordinates (x,y,z), the coordinates of (x_1,y_1,z_1) and (x_2,y_2,z_2) are given by:

$$\begin{aligned}(x_1,y_1,z_1) = (x + r_1\cos(\theta)\sin(\psi),\\ y + r_1\sin(\theta)\sin(\psi), z + r_1\cos(\psi))\end{aligned} \qquad (13)$$

$$\begin{aligned}(x_2,y_2,z_2) = (x + r_2\cos(\theta+\phi_1)\sin(\psi+\phi_2),\\ y + r_2\sin(\theta+\phi_1)\sin(\psi+\phi_2), z + r_2\cos(\psi+\phi_2))\end{aligned} \qquad (14)$$

where ϕ_1 and ϕ_2 denote the phase shifts, between the corresponding points (x_1,y_1,z_1) and (x_2,y_2,z_2).

Given n set of parameters with $i = 0,\ldots,n$, we define our three dimensional relational features, $\mathrm{R}(x,y,z,r_1,r_2,\phi_1,\phi_2,q)$, calculated on a local point (x,y,z) as follows:

$$\begin{aligned}\mathrm{RF}_i = [\mathrm{R}(x,y,z,r_1,r_2,\phi_1,\phi_2,q)]_i\\ \mathrm{RF}_i = \mathrm{rel}(X(x_1,y_1,z_1) - X(x_2,y_2,z_2))\end{aligned} \qquad (15)$$

Three dimensional invariant features in Eq. 4 becomes

$$\mathrm{IF}_{local}(i)|_{(x,y,z)} = \mathrm{hist}(f_i(x,y,z,\theta,\psi)X)), \tag{16}$$

where θ is an azimuthal coordinate running from 0 to 2π (longitude) and ψ is a polar coordinate running from 0 to π (colatitude). We end up with one dimensional feature vector with t bins.

IV. Results

For the evaluation of our three dimensional relational features, 100 training points from 6 CT datasets and 150 test points from 10 CT datasets were used for the classification task. These points are labeled as 1 (normal tissue), 2 (abnormal tissue) and 3 (others, i.e., blood vessels, kidneys that could not be removed during segmentation). The Support Vector Machine (SVM) classifier[2] was used for the classification task. Three tests were conducted in this work. As SVM kernel, we used the radial basis function (RBF) kernel for the classification task with a pair of parameters (C, γ). Here, C denotes the cost value that penalizes misclassifications and γ is the width-parameter in the RBF kernel.

We used four sets of parameters, $(r_1 = 0, r_2 = 5, \phi_1 = 0)$, $(r_1 = 2, r_2 = 3, \phi_1 = \pi/4)$, $(r_1 = 3, r_2 = 6, \phi_1 = \pi/2)$ and $(r_1 = 4, r_2 = 8, \phi_1 = \pi)$, each with $q = 20$. For all tests, we set $\phi_2 = 0$. This means only on longitude direction a phase shift was applied. The rel threshold, $\varepsilon = 0.196$ was used. At each point, we obtained $20 * 4 = 80$ features. Histogram relational features (hRF) were divided into 20 bins. The length of feature vector was reduced to 20.

For the first test, we set $C = 1000$ and $\gamma = 0.02$. hRF attains an accuracy of 86.67% for this test. Table I shows the confusion matrix with the true positive (TP) and false positive (FP) of the first test. 71 out of 79 are correctly classified as class 1 given 89.87% of TP and 7 out of 71 tissues from class 2 and 3 are incorrectly identified as class 1 given 9.859% of FP. The TP and FP for class 2 are 93.1% and 7.438%, respectively. However, the TP for class 3 is quite low (76.19%) since this class consists of several different tissues, which might have different tissue structures. Only 3.704% of tissues from class 1 and 2 are wrongly classified as class 3.

Table I
Confusion matrix and true positive (TP) and false positive (FP) for the three category classification task

%	1	2	3	TP	FP
1	**71**	6	2	71/79 (89.87%)	7/71 (9.859%)
2	0	**27**	2	27/29 (93.1%)	9/121 (7.438%)
3	7	3	**32**	32/42 (69.05%)	4/108 (3.704%)
total :				130/150 (86.67%)	20/150 (13.33%)

In the second and third tests, we compared the result of hRF with relational features that were computed by averaging. We refer to the features as aRF. We also compared the results with the gray-level co-occurrence matrix (GLCM) texture features [7] and Gabor wavelets [3]. We used the same SVM kernel as the first test for the classification. The GLCMs and Gabor wavelets used in this study were also extracted in 3D space.

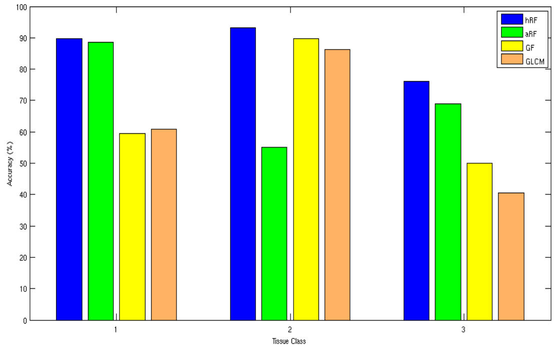


Figure 3. Comparison of the proposed texture features with 3D GLCM and 3D Gabor wavelets for the accuracy of different tissue classes.

Eight Haralick statistic measurements to describe the 3D GLCM were calculated using a window size of 5x5x5 in 13 directions($(0°, 45°)$, $(0°, 90°)$, $(0°, 135°)$, $(45°, 45°)$, $(45°, 90°)$, $(45°, 135°)$, $(90°, 45°)$, $(90°, 90°)$,$(90°, 135°)$, $(135°, 45°)$, $(135°, 90°)$, $(135°, 135°)$ and $(-, 0°)$) for contrast, homogeneity, angular second moment, entropy, maximum probability, energy and correlation measurements. 3D Gabor wavelets used in this study follow the formula given in [3]. A set of Gabor wavelets of different frequencies f_i and orientations (θ_j, ϕ_k) was calculated with the following representation:

$$\{\psi_{f_i,\theta_j,\phi_k(x,y,z)}, f_i = 0.5/(\sqrt{2})^i, \theta_j = j\pi/J, \phi_k = k\pi/K\} \tag{17}$$

A window size of 25x25x25 was used to produce Gabor filters and convolved with the same size of sub-image with point being evaluated was located at the center of the sub-image. The mean of convolution result was used to represent Gabor feature. Since a set of Gabor wavelets $\{\psi_{f_i,\theta_j,\phi_k}\}$ was used, IxJxK length of feature vector were obtained with $i = 0, ..., I-1$, $j = 0, ..., J-1$ and $k = 0, ..., K-1$. We set $I = J = K = 3$. This produced a total of 27 Gabor features (GF).

We performed the second test for comparison of classification accuracy for every class. On average, hRF outperforms GLCM by 26.67% and GF by 24%. aRF attains an accuracy of 80.67% in which hRF is better than aRF by 10%. See the graph in Figure 3 for comparison. It can be seen from the graph, the accuracy of GLCM and GF for class 1 and class 3 are worse than relational features. For class 2, aRF is the worst among all. However, hRF obtains the highest accuracy for all classes.

In the third test, we have computed the accuracy of

[2] http://lmb/lmbsoft/libsvmtl/svmtl.en.html

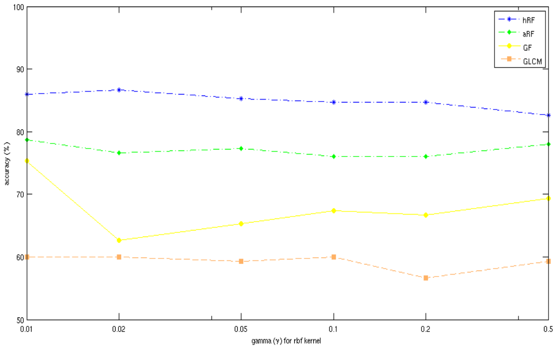


Figure 4. The accuracy of 3D relational features, 3D GLCM and 3D Gabor wavelets using different γ with $C = 1000$.

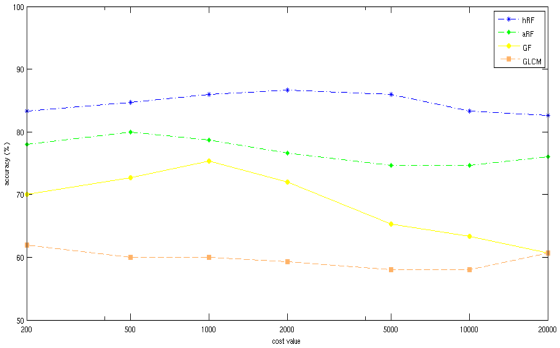


Figure 5. The accuracy of 3D relational features, 3D GLCM and 3D Gabor wavelets using different cost values, C with $\gamma = 0.01$.

all features using different pair of parameters (C, γ) for the RBF kernel. The result presented in Figure 4 was obtained by setting $\gamma = \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$ and $C = 1000$. In Figure 5, γ was set with 0.01 and $C = \{200, 500, 1000, 2000, 5000, 10000, 20000\}$. The results from both figures show that the relational features outperform GLCM and GF on the task. As predicted, hRF shows higher classification accuracy than aRF for all pair of parameters (C, γ) used in this test.

V. Conclusion and Future Work

In this paper, we proposed an extension of relational features to three dimensions. The purpose of this extension is to extract texture features from CT volumetric dataset to benefit from the full 3D information. The results from our study showed that the relational features achieved higher accuracy than the GLCMs and Gabor wavelets texture features in classifying different types of liver tissue from CT datasets. This indicated that, 3D relational features, especially hRF are promising for many medical applications that exploit texture features. For future work, we will consider nearby points to progress in the accuracy of liver tissue classification.

Acknowledgment

We would like to thank all the five anonymous reviewers for their helpful comments. We also immensely grateful to our colleague, Thorsten Schmidt for his comments and suggestions on an earlier version of this paper. The work has been partially sponsored by KPT/UMT under SLAI scheme.

References

[1] A. Jarc, P. Rogelj and S. Kovacic, *Texture Feature Based Image Registration* Biomedical Imaging: From Nano to Macro, 2007. ISBI 2007. 4th IEEE International Symposium on , vol., no., pp.17-20, 12-15 April 2007.

[2] A. Manduc, M. J. Carston, J. J. Heine, C. G. Scott, V. S. Pankratz, K. R. Brandt, T. A. Sellers, C. M. Vachon and J. R. Cerhan, *Texture Features from Mammographic Images and Risk of Breast Cancer*, Cancer Epidemiol Biomarkers Prev 2009, vol. 18(3), pp. 837–845, 2009.

[3] L. Shen and L. Bai, *3D Gabor wavelets for evaluating SPM normalization algorithm*, Medical Image Analysis, vol. 12, pp. 375–383, 2008.

[4] L. Tesar, D. Smutek, A. Shimizu and H. Kobatake, *3D Extension of Haralick Texture Features for Medical Image Analysis*

[5] M. Schael, *Invariant grey scale features for texture analysis based on group averaging with relational kernel function.*, Internal Report 01/01, University of Freiburg, 2001.

[6] M. Tuceryan and A. K. Jain, *Texture Analysis*, The Handbook of Pattern Recognition and Computer Vision (2nd. Eds.) by C. H. Chen, L. F. Pau and P. S. P. Wang (eds.), pp. 207–248, World Scientific Publishing Co., 1998.

[7] R. M. Haralick, K. Shanmugam and I. Dinstein, *Textural Features for Image Classification*, IEEE Transactions on Systems, Man, and Cybernetics SMC(6), pp. 610–621, 1973.

[8] S. Siggelkow, M. Schael and H. Burkhardt, *SIMBA - Search IMages By Appearance*, DAGM, LNCS 2191, pp. 9–16, 2001.

[9] T. Glatard, J. Montagnat and I. E. Magnin, *Texture based medical image indexing and retrieval: application to cardiac imaging*, Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval, pp. 135–142, 2004.

[10] T. Ojala, M. Pietikäinen and T. Mäenpää, *Gray scale and rotation invariant texture classification with local binary patterns*. In Proceedings of the 6th European Conference on Computer Vision, pp. 404–420, 2000.

[11] V. A. Kovalev, F. Kruggel, H. J. Gertz and D. Y. von Cramon, *Three-dimensional Texture Analysis of MRI Brain Datasets*, IEEE Transactions on Medical Imaging, vol. 20(5), pp. 424–433, 2001.

Highlighting the Essentials of the Behaviour of Reactive Systems in Test Descriptions Using the Behavioural Atomic Element

Lars Ebrecht
German Aerospace Center (DLR)
Institute of Transportation Systems
Braunschweig, Germany
lars.ebrecht@dlr.de

Karsten Lemmer
German Aerospace Center (DLR)
Institute of Transportation Systems
Braunschweig, Germany
karsten.lemmer@dlr.de

***Abstract* - The work described in this paper depicts an approach on how to close the lack of structure for test specifications, test descriptions and test data representations. On the example of a test data representation with the eXtensible Markup Language (XML), it will be shown how to form the structure using the behavioral atomic element and two testing patterns, i.e., for test stimuli and test reactions. The patterns are described using Petri net semantics. The scalability and flexibility of the approach, i.e., enabling the consideration of domain specific information, will also be shown on the example of the testing format. The pros and cons for using non-black box information in test descriptions and reference black box behavior specification for reactive systems, like actions, tasks and processes as well as system states in addition to the interface events, will be discussed in this paper at the end. The main achievement is a scalable, platform- and implementation-independent Test description using the meta-model described by the patterns.**

Keywords - Reactive systems; Validation and Verification, Real-time behaviour Pattern; Test Pattern; Validation Pattern; Test specification

I. Introduction

Black box test specifications mostly consist of stimuli and reactions mixed and alternated in any order according to the functional requirements to be covered. In some cases test steps are sporadically inserted in order to indicate actions, processes or changes inside the system under test (SUT), e.g., here a safety-critical distributed reactive system - a train control unit. Concerning the test creation and evaluation the practice shows that it is very useful to add this information. These non-black test steps enable a better understanding of certain situations as well as figuring out specific system features and functionalities. Consequently, no order or any order concerning the use and mixture of stimuli, reactions, internal processes and system states of a reactive system is fixed by the application of the behavioral atomic element as generic pattern for the behavior of reactive systems and black box tests. Further it will be shown that the behavioral atomic element does not restrict the test description too much. It creates moreover a scalable and concerning the consideration of domain specific information, flexible as well as consistent and transparent the behavior description for reactive systems. The introduction and demonstration of the approach is done on the example of a test description for a train control onboard unit.

A. *Context of the work and application example*

The test description that is used as application example in this paper, addresses the European Train Control System (ETCS). ETCS comprises two main safety-critical distributed real-time components, i.e., the ETCS onboard unit (OBU) located on each train and the Radio Block Center (RBC) on the trackside. The RBC mainly supervises the location of the trains for a certain area of a track in order to provide movement authorities to the trains in that area. The OBU uses the information got from the RBC mainly to supervise the train movement, i.e., the maximum allowed speed and distance allowed to travel. All the functional requirements telling how and when to do what are specified in the European wide standardized System Requirement Specification (SRS) (Subset-026 [5]). In order to check if an OBU fits the SRS, i.e., the behavior, all the interaction and way of exchanging information via different physical interfaces with the track, the functional requirements have been analyzed and assigned to features and test cases. The test cases have been selected, parameterized and combined to form virtual train trips, called test sequences, in order to emulate the start of mission, train movement with certain specific conditions as well as the end of the virtual trip. Thus, the technical conformity and interoperability of the OBU will be checked against SRS (Subset-076 [6]).

The conformity and interoperability tests of Subset-076 are used in independent laboratories like the Railway Simulation and Testing laboratory (RailSiTe® [14]). Only giving some rough figures to the modular and distributed test environment, the RailSiTe contains different modules for the simulation of train dynamics and track properties as well as different physical black box hardware interfaces, i.e., high frequency signals, digital I/O, TTL, GSM-modem-serial-connection, camera and robot for a touch screen display acting as man machine interface of the OBU [1].

Concerning the tests applied in this test environment there is a separation of the logical and functional behavior on the one hand and the physical interface behavior, i.e., the signal generation and emulation, on the other hand. The test and reference description represent the logical and functional behavior that are used by different interface modules to encode the information and to build the physical signal in order to stimulate the SUT, i.e., the OBU. The emulated train dynamics, i.e., the acceleration and braking, are logically and mathematically described in the test and reference behavior description.

B. Structure of the paper

This paper is structured as follows. Section II depicts related work and the differentiation of the approach described in this paper to other present approaches for the test specification and behavioral description for reactive systems. The following Section III introduces the atomic element for the behavior of reactive systems and the two testing patterns, i.e., for test stimuli and test reactions that are described using Petri net semantics [13]. Section IV represents the main section of the paper introducing the structure of the test description and test data representation with the eXtensible Markup Language (XML) [8] followed by the demonstration of scalability and flexibility concerning the consideration of domain specific information. The last Section V will discuss and conclude the presented approach and results.

II. RELATED WORK

There exist various ways and standards for modeling and describing the behavior as well as tests for reactive systems. Beside the continuous signals in the electrical engineering or control theory domain, there are several standards coming from the software engineering domain, e.g., the Unified Modeling Language (UML), the System Modeling Language (SysML), the UML Testing Profile (UTP), the Testing and Test Control Notation (TTCN-3). The approach presented in the following does not touch continuous signals in the way they are described by Matlab / Simulink, LabView or similar tools or approaches.

The presented approach is more related to the formal mathematical and software engineering domain. In this domain the main standards are listed above. The UML [3] and SysML [2] specify how to describe systems and processes within a user-friendly representation in form of different diagrams with certain associated graphical symbols and elements. Both do not include a clear and formal concept for the behavioral description of reactive systems. More or less activities, operations, message exchanges and states are spread over the Activity, the Sequence and the State diagram without any concrete order and relationship specific for the reactive system behavior.

The UML Testing Profile (UTP) enables the description of black box tests [11]. You can describe test contexts, test configurations and test components. For the behavioral part of a test description the UTP offer test cases, test case parameters, stimuli, observations and test data as well as test routines and test procedures [11]. However, there is no comparable approach of a Meta model like it is defined by the behavioral atomic element. There are pre- and end-conditions of operations and invariant system properties without any concrete mapping to figure out the relation to the behavior of a reactive system as well as having an atomic element.

The Testing and Test Control Notation (TTCN-3) of the European Telecommunications Standards Institute (ETSI) is an additional standard for the black box test description. It mainly focuses on the implementation, execution and evaluation of black box tests [4][12]. The semantics of flow graphs, especially the flow graph frame that consists of start node, basic node and an end node [4], are not consequently used as generic basic concept for the behavioral description of reactive systems like it will be shown in the following with the atomic element.

III. THE BEHAVIOURAL ATOMIC ELEMENT AND RELATED TEST PATTERN

In the following, the behavior of a reactive system is understood and described by the behavioral atomic element [10]. The element comprises the system configuration, i.e., system states (S_i), in- and out-going events (e.g., periodical, synchronous and asynchronous events) (E_j), sequential and parallel activities, operations or processes (A_k, AS_k).

In the middle of Figure 1. the pattern of the atomic element is depicted using the Petri net semantics [13]. The Petri net representation indicates the activity A_k as transition encapsulated by at least one in-coming event E_i, potential out-going events E_o as well as the starting state S_s and end state S_e of the functionality. The events and states are shown as places. The “b” represent a condition that enable the specification of further trigger conditions, e.g., temporal conditions, for the activation of the activity A_k. This is called the compact representation of the atomic element.

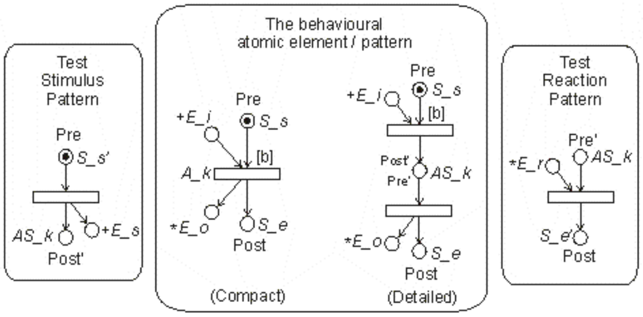


Figure 1. Basic Patterns describing the atomic behaviour of reactive systems, test stimuli and test reactions

Figure 1. shows a second detailed and unfolded representation of the atomic element that depicts the activity AS_k as place. In comparison to the compact representation the unfolded one highlights the activity. So, the activity is indicated as a meta-state, i.e., the system configuration might change or might be in-between two stable system states also having the possibility of further investigations concerning the description of in-coming and out-going events. Thus, the unfolded representation of the atomic element emphasizes the system activity enormously. The whole configuration enforces the understanding of the system behavior by relating events together with states and the activity to a specific functionality.

The consideration of system states, events and activities improve the transparency and understandability of the system behavior. Even if this seems to be like a grey box test, this is not the case due to the fact testing the SUT via the black box interfaces and without referring to internal interfaces or structures like sub-classes or modules. On the very left side of Figure 1. the pattern describing the events stimulating the SUT (E_s) is shown in comparison to the pattern of managing reactions (E_r) coming from the SUT on the very right side of Figure 1.

Time is considered as a global discrete variable in addition to system states, event occurrences and process durations. Of course concerning non discrete real-time you will have to consider the WKS-Sampling-Theorem as well as accuracy aspects like jitter with proper tolerance borders and intervals.

A. Test scenario defined by the atomic element and the testing and validation patterns

For a better understanding one very easy scenario demonstrates the use and interaction of the three patterns mentioned before concerning a common situation, i.e., testing a SUT with a certain test environment (TE) (see Figure 2.). In the middle there is a UML [3] sequence diagram showing the TE and the SUT. From the TE the stimulation E_s is triggered stimulating the SUT in the sense of a certain in-coming event E_i. After the reception of this event the operation A is started maybe emitting the result or confirmation or response with an outgoing event(s) E_o that is taken into account as reaction E_r from the SUT. In the background the Petri net representation is shown separated in TE side on the left and the SUT side on the right.

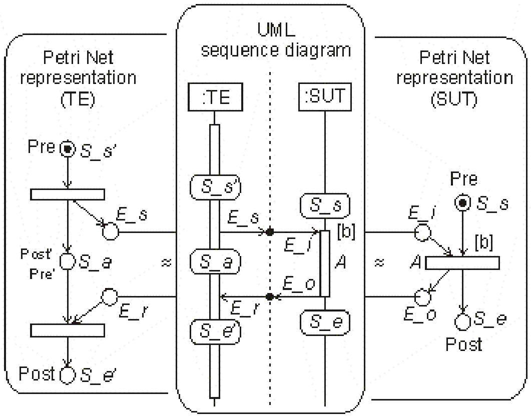


Figure 2. Test scenario: Test environment (TE) stimulating the system under test (SUT) in order to check function A

IV. APPLICATION OF THE ATOMIC ELEMENT TO THE TEST DESCRIPTION

In the previous section, the behavioral atomic element has been described that define the functional behavior of a reactive system as well as the other two test and validation patterns for the stimulation of a SUT as well as for receiving reactions from a SUT, here a train control unit (OBU). Now, it will be demonstrated how these patterns are used to form a test description and reference behavior on the example of a test data representation by an XML-schema [7]. The XML-schema defines the structure of an XML [8] file assigned acting as instance of that schema.

In the first subsection, the overall structure and content will be roughly depicted. The second subsection shows the application of the behavioral atomic element in detail. The last two subsections explain how the patterns enable a scalable and customizable use of the test description.

A. Overview and structure of the test description format

The highest level of the overall structure includes four elements (General, StartingConditions, StepList and EndConditions). The element General contains the ID of the test sequence, a title, description, a substructure for the test sequence release and modification history table beside a link to a set of features and test cases. Out of the set of test cases some will be selected, instantiated and concatenated with others in a test sequence. Accordingly specific starting conditions will be fixed and stored in the StartingConditions element. As shown in Figure 3. this item comprise two generic and two domain specific sub-elements. The Sub-element Variable and Set allow defining variables and sets of variables with concrete values, intervals or enumerations in order to describe the system state and system variables. M_MODE and Location represent customized entries for the mode of an ETCS onboard unit (OBU), e.g., no power, standby, full supervision, shunting, etc., and the other comprises several variables defining the train location, i.e., the orientation of the train and travelled distance in relation to certain waypoint in the track. There exist further application domain specific entries in the starting that are not mentioned here.

The EndConditions element is similar to the StartingConditions element and describes accordingly the system state reached at the end of a test sequence. Between the starting and end conditions there is the StepList element located. This element takes the main role in the overall structure. As displayed in the Figure 3. it contains a list of steps that can be one of three following kinds, i.e., a state, an action event or an action. Besides, each step has a unique step ID and can be grouped, e.g., in the case of test case associations or sequential or parallel ordered steps. In Subsection IV.D it will be explained more in detail how the group element can be used to describe conditions and relationships over different steps for domain specific aspects and dependencies. Last but not least you can define variables in the SignatureVariableList that interconnect steps in order to exchange data among different steps.

The three kinds of the step as main element mentioned before are used to describe the behavior as defined by the atomic element. For each step it has to be chosen if the step describe a system state or an action event, i.e., a stimulus to be triggered by a test environment or a reaction coming from the SUT, or an action that is assumed to be active in the SUT. In the following section it will be explained how these three different elements will be connected in order to satisfy the behavioral atomic element.

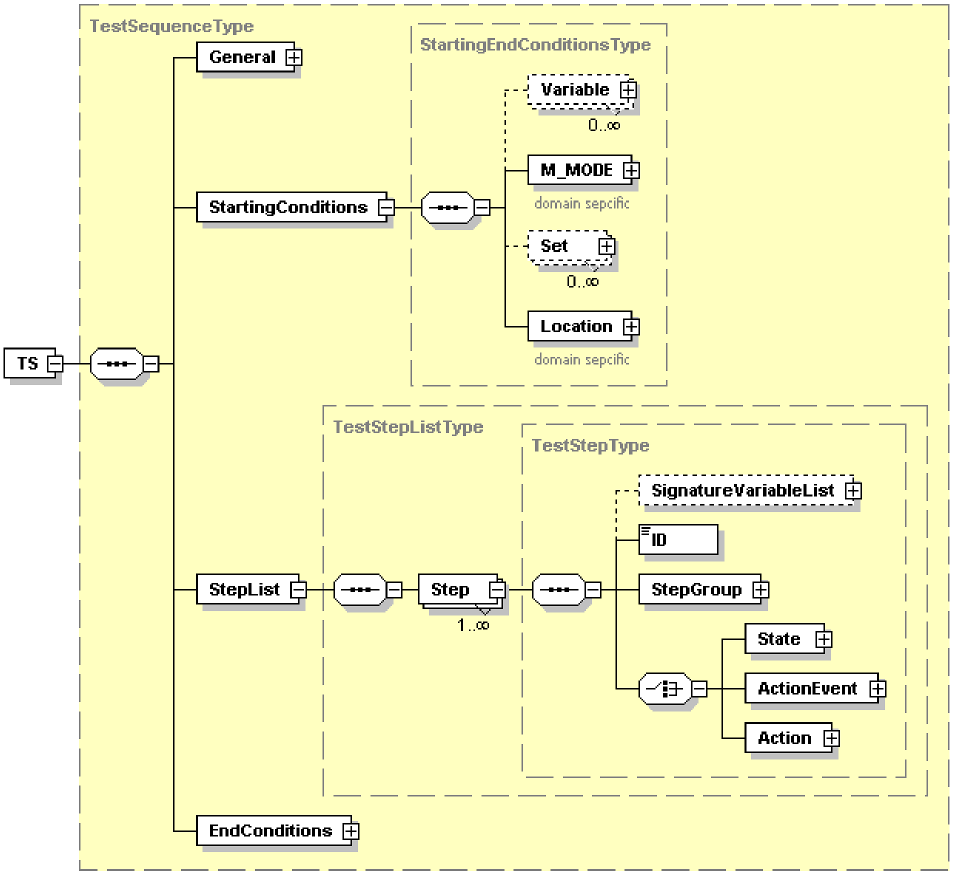


Figure 3. Overal structure of the test data representation defined by an XML-Schema [7]

B. *Details of the test data description according to the behavioural atomic element*

The beginning of a function should be a system state. As depicted in Figure 4. a state is defined by several variables contained in the VariableList substructure or by sets of variables in the Set element similar to the starting and end conditions described before. Furthermore a state has an identifier (ID), a name and a textual annotation.

A State is followed by an action event, i.e., one or more in-coming events stimulating the SUT. Action events are divided in in-coming, i.e. stimuli (E_i) and out-going events, i.e., reactions (E_o). All action events are described by an ID, a name, the event direction and interface, a description and related data (see Figure 4.). The Data sub-element comprises two alternatives, a reference to another action event and a set of elements composed of the action event trigger, duration, delay, function name and a function parameter list. The first alternative is used for the case if a step specifies for instance an optional packet that should be included in an event together with other packets, which are specified by other steps. In this case it is possible to include this packet together with the other ones in one action event whereby the data are only included in one step and the other refer to this action event. The second alternative represents the usual case where an event is triggered by, e.g., the location of the train, the time elapsed from the beginning of the simulation, an other action event or a set of variables defining a certain system state. The Duration and the Delay element contain a variable type describing the time for the generation and transmission and possible delays related to the trigger point. The function-name is used to determine runtime commands or routines of the test environment using all the variables and its values contained in the element ParameterList.

The last of the three main elements, the Action, depicts an action or task of the SUT. It is similar to the ActionEvent element. It has an ID, name and description and contains a substructure for the related data. Except from the ActionTrigger element the Data element is equal to the one of the action event. In comparison to the action event the action is triggered by one or more in-coming events that have to be received before.

Due to the fact that you want to violate the specified rules and conditions of a behavior in some cases testing a SUT, it is not forbidden to mix the three main elements also against the behavioral atomic element what represent an abnormal situation. You can ever check and identify these violations applying checks to the test and reference behavior description finding deviations in relation to the behavioral atomic element in order to check if you wanted to do so or not.

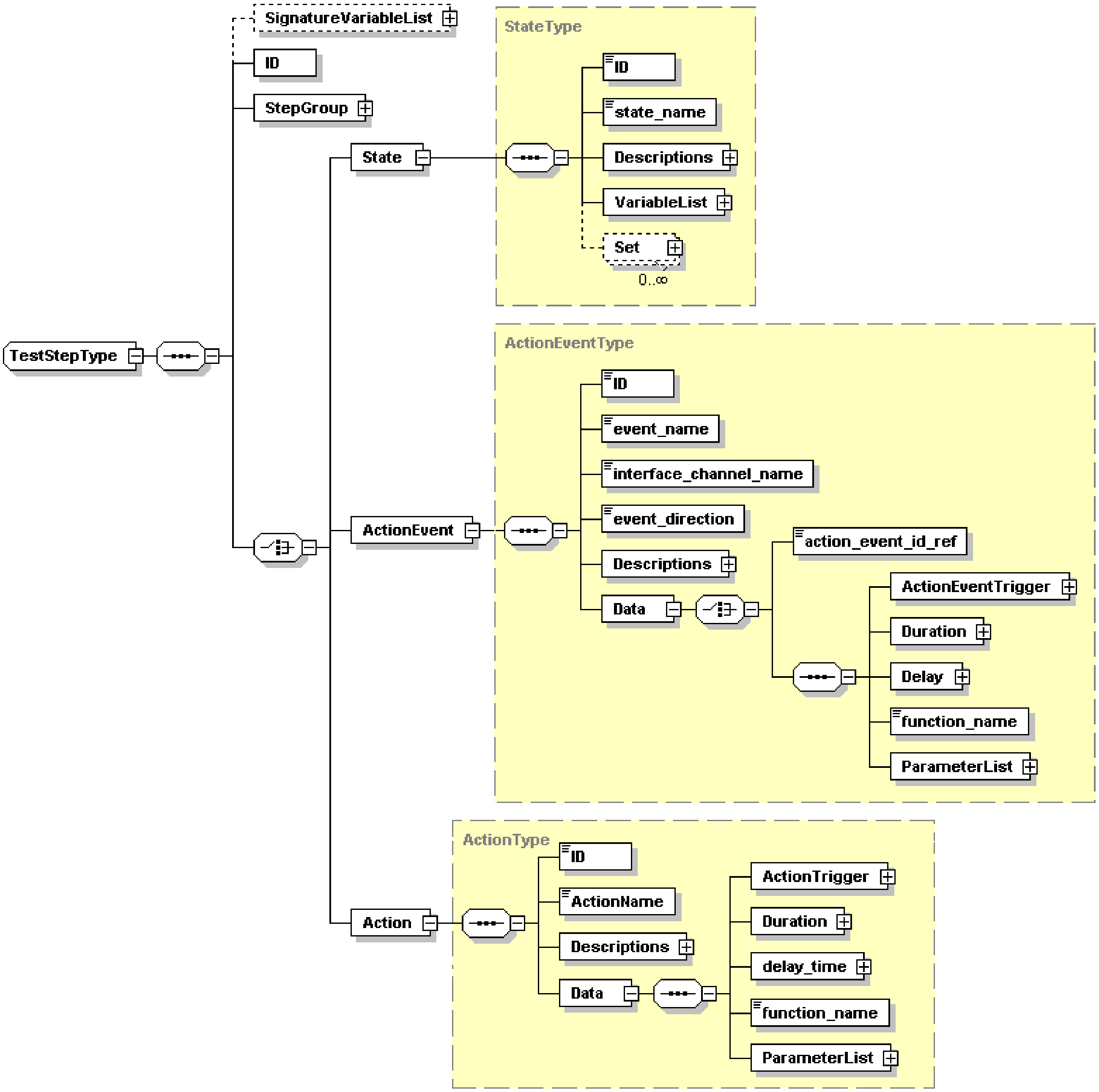


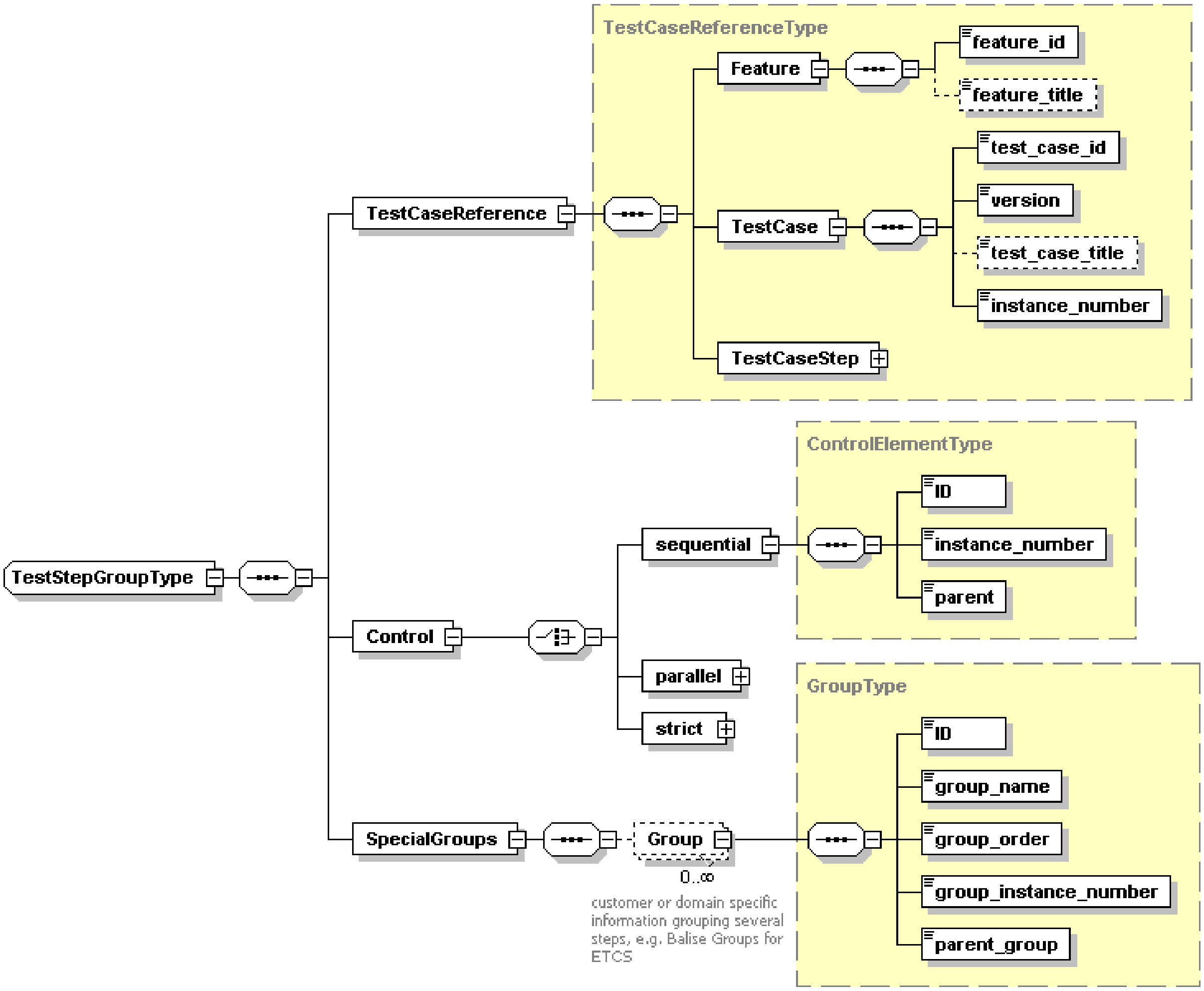
Figure 4. The atomic element builds the generic structure for a test and reference behaviour description

C. Scalability of the behaviuor description

The application of the atomic element enables on the higher abstract level as well as on the lower detailed level the possibility to combine test sequences with matching starting and end conditions [9], this not very astonishing, but also for concrete functionalities according to matching of post and pre states (compare with Figure 1.). So, small functional units can be consistently combined, i.e., an action belonging to a certain functionality including its trigger event(s) as well as the starting and end state. Test cases only containing starting and end conditions and black box message passing in between having a lack of building sup-units for smaller function inside bigger ones like the atomic element cramping events and an action by system states. Thus, there is the same consistent approach for test sequences over test cases until the steps included in a test case.

D. *Differentiation of the atomic element considering domain and user specific relations and dependencies*

The most popular group for test steps is the association to a test case. Black box stimuli and reactions are related to test cases and these for itself to features in order to manage the system complexity in principle. Beside the step order like sequential, parallel or strict execution or occurrence defined in the Control element (refer to Figure 5.), there is the need to specify domain and user specific relationships and dependencies. In the context of testing an OBU there is the need to describe a logical order for a certain type of events, i.e., Balise messages. One Balise message contains one up to eight Balise telegrams emitted by a passive discrete device that is in principle comparable with RFID tags only that additionally Balises include some more safety mechanisms in order to avoid copying telegrams. A Balise message has to be consistent, that means that the telegrams have to receive in the right and continuous order and completely, i.e., all of the telegrams. This association is described by the Special-Groups element. This element comprises an ID, name, group order, parent group for hierarchical dependencies and the group instance, in case that there more than one group occurrences.



customer or domain specific information grouping several steps, e.g. Balise Groups for ETCS

Figure 5. Vertical differentiation of the test steps

V. Conclusion

The contribution introduced the behavioral atomic element described as Petri net acting as pattern for a Meta model and forming a test data representation and reference behavior description. Beside, two other testing and validation patterns complete the scenario testing a reactive system with a certain test environment on a logical abstract level. Mostly, test descriptions and specification consider more or less mainly test stimuli and reactions of a system under test. The presented approach does not only depict the application of the atomic element to a test data representation, it also figures out how this reference behavior description can cope with domain specific requirements and dependencies as well as enabling a scalable, consistent and transparent reference behavior description highlighting the essentials of the behavior of a reactive system, i.e., system states, in-coming events or trigger events, actions or operations and out-going responses. Thus, the consistency and transparency of the test specification for any reactive system is improved by the definition of proper behavioral units and functions with the help of the behavioral atomic element. The presented XML format defined in the Schema is currently used to automate the test execution and evaluation in the rail simulation and testing laboratory RailSiTe®.

ACKNOWLEDGEMENT

Many thanks to the group rail technology, especially to C. Preuk and C. Torens for the feedback and review comments.

REFERENCES

[1] C. Torens and L. Ebrecht, Remote Test: A Framework for Testing Distributed Systems, Proc. International Conference on Software Engineering Advances (ICSEA2010), Aug. 2010.

[2] Object Management Group, „OMG Systems Modelling Language“, Ver. 1.2, June 2010, www.omgsysml.org/#Specification (23.08.10).

[3] Object Management Group, „Unified Modelling Language“, Superstructure Specification, Ver. 2.3, May 2010, www.omg.org-/spec/UML/2.3/ (23.08.10).

[4] ETSI, ES 201 873-1, „Testing and Test Control Notation - Part 1: TTCN-3 Core Language,“ Ver. 4.2.1, May 2010, webapp.etsi.org-/workprogram/Report_WorkItem.asp?WKI_ID=29256 (23.08.10).

[5] ERA, „ERTMS/ETCS – Class 1, System Requirement Specification Subset-026 (SRS)“, Version 2.3.0d, 21 Apr. 2009.

[6] ERA, ERTMS/ETCS – Class 1, „Technical Specification for Interoperability: Subset-076-6-3 (Test Sequences)“, Version 2.3.1, 19 Oct. 2009.

[7] XML Schema, W3C, http://www.w3.org/XML/Schema (23.08.10).

[8] T. Bray et al., W3C, XML, www.w3.org/TR/2008/REC-xml-20081126, 26 November 2008 (23.08.10).

[9] L. Ebrecht, M. Meyer zu Hoerste and K. Lemmer, “The Basic Concept for the Formal Test Description - Horizontal Composition and Vertical Differentiation of the Atomic Element,” Proc. Formal Methods for Automation and Safety in Railway and Automotive Systems (Forms/Format), GZVB, Jan. 2007, pp. 447–457. ISBN 13: 978-3-937655-09-3.

[10] L. Ebrecht and K. Lemmer, “Konsistente Verknuepfung von Aktivitaets-, Sequenz- und Zustandsdiagrammen - Darstellungsunabhaengige und formale Semantik zur Verhaltensbeschreibung von Echtzeit-Systemen,” Proc. Mobilitaet und Echtzeit - PEARL. Informatik Aktuell. Springer, Dec. 2007, pp. 49–58. ISBN 978-3-540-74836-6.

[11] Object Management Group. (July 2005). UML Testing Profile, Ver. 1.0-05-07-07, www.omg.org/technology/documents/formal/test_profile.htm (23.08.10).

[12] J. Grabowski. TTCN-3 - A new Test Specification Language for Black-Box Testing of Distributed Systems. In: Pro-ceedings of the 17th International Conference and Exposition on Testing Computer Software (TCS 2000), Washington D.C., www.swe.informatik.uni-goettingen.de/publications/JG/Grabowski.pdf, Jun 2000 (23.08.10).

[13] Billington, Jonathan ; Reisig, Wolfgang: *Application and Theory of Petri Nets*, Springer, 1996, ISBN 3-540-61363-3

[14] RailSiTe DLR homepage, www.dlr.de/fs/en/desktopdefault.aspx/-tabid-1235/1688_read-3254/ (23.08.10)

CUX Patterns Approach: Towards Contextual User Experience Patterns

Marianna Obrist, Daniela Wurhofer, Elke Beck & Manfred Tscheligi

Christian Doppler Laboratory for "Contextual Interfaces"
HCI & Usability Unit, ICT&S Center
University of Salzburg
Salzburg, Austria
Email: {firstname.lastname}@sbg.ac.at

***Abstract*—User experience (UX) is highly influenced and even changed by the context in which it occurs. So far, both concepts "user experience" and "context" have been discussed a lot to various extent and in different dimensions. Within this paper, we aim to bring these two important areas closer together by using patterns. We introduce the contextual user experience patterns (CUX patterns) approach. Precisely, we argue for using patterns to describe knowledge on how to influence the users' experience in a positive way by taking context parameters during the interaction with a system into account. To do so, we provide a detailed description of how to structure CUX patterns, referring to the context "car" as one of the two main application areas, which is investigated in our recently established laboratory on contextual interfaces.**

***Keywords*-user experience; context; contextual user experience; patterns approach; patterns structure; contextual interfaces; car application area**

I. Introduction and Motivation

In the field of Human-Computer Interaction (HCI) and related fields, it is increasingly recognized that apart from standard usability and ergonomic principles, the much broader concept of user experience (UX) needs to be considered intensively to design next generation interaction innovations [1]. This includes aspects such as fun, enjoyment, emotion, sociability, and other factors.

Moreover, there is an observable trend in HCI towards novel and alternative forms of interaction, moving away from traditional desktop computing to computing in various contexts with various interfaces. One general form of novel interfaces are so-called "contextual interfaces" (see [2][3]), which are designed according to the needs and behaviors of people in specific contextual situations and have the potential to be conceptually as well as technically adapted to the characteristics of the specific context. Thus, an in-depth understanding of the particular context is needed for designing contextual interfaces.

In addition to knowledge on the context, there is a need for insights into how a user perceives an interaction with a system. Thereby, evoking a positive feeling within the user through the usage of the system increases the potential for re-usage of the system. We thus claim that enabling the user a positive experience by considering the context parameters during an interaction is one important ingredient for the success of contextual interfaces. However, as far as we are aware, there are no such insights and best practices available yet for contextual interfaces.

So far, both concepts "UX" and "context" have been discussed a lot to various extent and in different dimensions. We aim to bring both concepts together to reach a more comprehensive understanding of contextual UX [3], which opens up different roads for research and challenges for the HCI community in all design and development phases. In particular, we bridge the two concepts by using the patterns approach. Patterns exist in many areas, ranging from architectural patterns (e.g., [4]) to patterns for human-robot interaction (e.g., [5]). However, despite the growing importance of contextual interfaces as well as UX, the patterns approach has not been applied yet for contextual UX design. Within this paper, we introduce "Contextual User Experience Patterns" (in short, CUX patterns). We propose a pattern structure and describe each part, but without giving a detailed description of CUX patterns themselves.

Building on our previous work on UX patterns for audio-visual networked applications [6], we again use the patterns approach for describing knowledge on how to influence the users' experience in a positive way when interacting with a contextual interface. CUX patterns represent a pattern collection for documenting and collecting best practices in the area of contextual user interfaces. Thus, developers and designers can be supported in producing high-quality user-centered applications. This research represents a main element of a recently started Christian Doppler Laboratory (CDL) on "contextual interfaces" at the University of Salzburg, Austria.

The present paper is organized as follows: Section II explains the background of our research on contextual interfaces and section III summarizes related work on (HCI) patterns, as well as clarifies the potential of patterns for contextual user experience. In section IV, the main idea of CUX patterns, and their structure are described and summarized in the final section.

II. Research Context and Background

Within this section, we shortly explain the background of our research on contextual interfaces (basic research within our laboratory) and our understanding of UX factors and context parameters.

A. Laboratory on Contextual Interfaces

The overall goal of the laboratory on "contextual interfaces" is to further strengthen existing research and in particular to gain new insights into the multidimensional aspects of contextual interaction. Contextual interaction can be considered as situated human-computer/machine interaction, which is dependent on a multitude of factors. We investigate contextual interaction from a constructional and methodological viewpoint to develop a deeper understanding of optimal contextual user experiences and their influences.

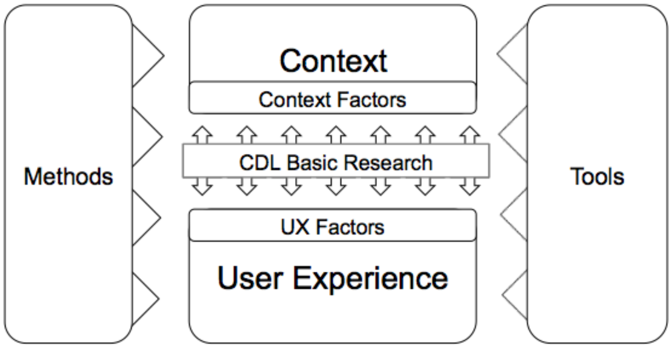


Figure 1. Basic Contextual Research Parts of our Christian Doppler Laboratory (CDL): Bridging User Experience (UX) Factors and Context Factors by doing Basic Research Activities on Methods and Tools.

The planned work is divided into two parts (see Figure 1). On the one hand, UX factors and context factors are investigated, and on the other hand, methods and tools to study contextual interfaces are addressed. Thus, these basic research parts will deliver a foundation for the second research part which addresses two specific application areas, namely car and factory. Both parts are related to each other and aim to increase the knowledge on contextual user experience. One main outcome of this research project (for the next seven years) are more generalizable UX patterns for contextual interfaces, i.e. CUX patterns. First of all, deeper insights on the main elements are needed and are shortly introduced in the following section.

B. Understanding User Experience and Context

The concept of seeing technology in terms of experience was originally introduced by McCarthy and Wright [7] and further extended by several attempts, models, definitions etc. (e.g. [8][9]). In our previous research we already investigated UX in different application areas (i.e. human-robot interaction and audiovisual networked applications) and identified a set of relevant UX factors to understand the aspects that make an experience more successful and valuable to users and contribute to positive contextual user experiences.

A small selection of these UX factors, which we conceive as relevant for the context car and factory, is provided below:

- Emotions (e.g., What emotions does the interaction with a system provoke?)
- Fun/Enjoyment (e.g., Do people enjoy interacting with a system?)
- Co-Experience (e.g., How do other people influence the experience?)
- Trust (e.g., To what extent do people trust a system?)
- Feeling of Security (e.g., How save do people feel?)
- Comfort (e.g., What is perceived as comfortable?)
- etc. (e.g., motivation, added value, engagement, etc.)

Based on empirical insights, the factors listed above will be further extended.

Moreover, in HCI, several definitions of context have been proposed during the last years. One of the most complete definitions of context, which we use as a starting point in our research, is provided by Dey [10]: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Overall, a common understanding of context is not fully developed up to now. Especially the influence and potential of context parameters for designing interaction innovations is poorly conceived and underpins the need for empirically grounded research. However, some relevant context parameters can already be extracted from [11] and are summarized as follows:

- Physical context (e.g., spatial location)
- Task context (e.g., task types, interruptions)
- Social context (e.g., presence of other people, culture)
- Temporal context (e.g., duration, time of day, week, and year)
- Technical and information context (e.g., other systems and services)

Overall, the focus on contextual user experience is not completely new in the history of patterns. The first who emphasized a focus on the human perspective was Alexander [4]. He argues that it is important to investigate how people experience architectural constructs, and to take the user's experience into account when constructing new buildings. Other authors (e.g., [12], [13], [14], [15] and [16]) also argue to consider human activity and experience and thus underpin the relevance of this topic.

The advancement of our approach is the strong combination of the two concepts "UX" and "context". Furthermore, our approach bases on empirical evidence from the two context application areas car and factory (see section IV-B).

III. Related Work

Within this section, we will provide more insights on the idea of patterns and their application in the area of HCI as well as their potential for addressing contextual user experience.

A. The Idea of Patterns

Patterns exist in many areas. The concept of patterns was first described by Alexander [4] who developed more than 250 patterns showing best practices and thinking in architecture. Later, computer scientists captured the idea of patterns and successfully applied it to common problems in software engineering [17]. Over the past years, the area of HCI also adopted the idea of patterns for conveying principles and best practices of good interface design (e.g., [18][19][20][21][22]).

The concept of interaction design patterns is known under different names such as interaction (design) patterns, user interface (UI) patterns, usability patterns, web design patterns, workflow patterns or, less precisely, HCI patterns. In general, these patterns share a lot of similarities and all provide solutions to common usability problems in interaction and interface design. For a detailed review on patterns in HCI we refer to Dearden and Finlay [23].

As the wide usage of patterns shows, patterns have proven to be an effective tool for designing usable systems. The idea of “reusable” solutions for recurring problems is important for both novice and experienced designers.

B. Characteristics of Patterns

Already in 1977, the main characteristics of patterns were pointed out by Alexander [4], stating that “each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over”. Newer definitions of patterns define them as “structured textual and graphical descriptions of a proven solution to a recurring problem” [24], as “tools for capturing useful design solutions and generalizing them to address similar problems” [12], as “devices for re-use, generalisation and design” [25], or simply as “descriptions of best practices within a given domain” [20]. These definitions illustrate one of the main characteristics of patterns – the characteristic of reusability.

Another characteristic of patterns is their context, i.e. the physical and social environment the pattern is embedded in. Furthermore, patterns represent a systematic approach to design which can be beneficial for a wide audience of practitioners [26] and thus can be considered as an “effective knowledge management tool” [16]. Patterns provide a collective vocabulary [27] or lingua franca [28] and thus facilitate communication between different stakeholders.

C. Special Application Areas of Patterns in HCI

Patterns are adopted in a variety of application fields. Recently, Zimmerman [29] used design patterns for describing how to apply the product attachment theory to the interaction design of a product. Kahn et al. [5] used patterns in the area of Human-Robot Interaction for explaining how to achieve sociality in Human-Robot Interaction. Patterns have been adopted for supporting innovative game design [30], for describing best practices in ubiquitous computing applications [31] and in the design of social interfaces [32], as well as for teaching HCI principles [33].

According to Zimmerman [29], patterns can also be used as a method for analyzing user research. This is illustrated, for instance, by Martin et al. [34] and Crabtree et al. [35], who use patterns for organizing and presenting ethnographic material. A recently published book targets the design of social interfaces [32]. Principles and patterns for social software are introduced by giving practical design solutions for improving the interfaces of social websites. In our recent research, we also successfully explored the patterns approach for audiovisual networked media (social media applications) by introducing UX patterns (see [36][37]).

D. User Experience Patterns

Designing for a good UX is an increasingly important topic in academia and industry (see [38][39]). In our previous research, we developed 30 UX patterns for audiovisual networked applications [37] based on a huge range of collected empirical data, which was further categorized into main UX problem areas. These areas cover the main UX issues in audiovisual networked applications. As advancement of this research, we intend to extend the application areas beyond audiovisual applications towards the application area car and factory. Thereby, we additionally want to strengthen the importance of context factors (the influence of contextual parameters) on the UX.

The multifaceted adoption of patterns illustrates their flexibility as well as their potential for storing and representing knowledge. We are convinced that the patterns approach can be further strengthened by putting a strong emphasis on the context and relevant context parameters for an application area, linked with relevant UX factors.

IV. The Potential of Patterns for Contextual User Experience

Within this section, we provide a structured overview on the details of our CUX patterns approach, including a definition as well as our strategy for structuring the patterns (see CUX Patterns Structure). Therefore, we selected the application area “car” as a case example to clarify our approach.

A. Details on the CUX Patterns Approach

Best practices for designers and developers are needed to guide the development of contextual interfaces which, for instance, support user's trust and feeling of comfort. We suggest to transform such best practices into CUX patterns for contextual interfaces. In the following, we focus on "car" as an application area. We suggest the following initial list of UX demands (exemplary problem areas), which does not strive for completeness, and needs to be extended based on empirical insights:

- Trust: How to increase the user's trust in the car when interacting with in-car interfaces?
- Comfort: How to give the user a feeling of comfort when using in-car interfaces?
- Personal and social benefit: How to raise the personal and social benefit of using in-car interfaces?
- Feeling of Control: How to raise the users' feeling of control when using in-car interfaces?

For these UX problems, the CUX patterns for the application area car should provide proven solutions. One problem area can comprise more than one pattern, depending on how many different best practices exist for the problem addressed. The patterns subsumed under each problem area should address a more specific problem (which can be subsumed under the corresponding problem area) and provide detailed solutions for this problem.

In general, CUX patterns provide solutions on how to improve a user's experience when interacting with a contextual interface in a specific application area. More specifically, CUX patterns are characterized by combining two aspects, i.e. (a) the application area of the patterns including the most relevant context parameters and (b) the specific (recurring) UX problem (demand) which the pattern intends to solve. Thus, CUX patterns can provide useful support during the design and development phase of an application. They especially provide guidance in the initial steps of an experience-centered design approach, where the designer does not yet have a clear picture on the potential users and their expectations, that influence their experience with a new system. Thus, it is relevant to consider CUX patterns quite from the beginning as an additional pool for inspiration.

B. CUX Patterns Structure

As a starting point for developing the CUX patterns we build on the structure of Van Welie [16] and Borchers [38] and extend it in particular regarding UX factors and context parameters. Context is currently limited to the characteristics of the usage context for which the pattern can be applied, but does not link the context/context parameters back to the UX factors relevant for the particular application area. In the following, the suggested structure is described and illustrated by the exemplary pattern "Feeling of Security" for the application area car.

- *Name/UX Factors:* The name of a pattern is essential. It should describe the main idea of the pattern in one or a few words; it should be both descriptive and unique so that it helps in identifying and referring to the pattern. In other words, patterns should be easy to remember and communicate. Moreover, the naming strategy should be consistent across all patterns collected, also supporting better memorization and easier communication (about patterns). Finally, the user experience factors that are addressed by the pattern should be included along with the name in combination with the relevant context parameters. *An example for a CUX pattern name in the application area car can be: "Feeling of Security".*
- *Problem:* The problem states the major issue the pattern addresses, formulated as a question based on the experiences made by the users. Problems in CUX patterns are formulated from the users' perspective and are related to the usage of the system. *An example for a CUX pattern problem in the application area car can be: "How to raise the users' feeling of security when using the speed control?".*
- *Forces:* The forces further elaborate the problem statement. They depend on the application area and can describe various trade-offs, constraints, or concerns related to the use of the pattern. *Examples for CUX pattern forces in the application area car can be: "The user wants to see the status of the speed; the user wants to change the speed, etc.".*
- *Context Parameters:* The context section of a pattern should describe when it is appropriate to apply a particular pattern, giving information about the characteristics of the context of use, including the context parameters listed above for which the pattern can be applied. *An example for a CUX pattern context parameter in the application area car can be: "Use this pattern when you want to support the users' feeling of security while driving"*. The context parameter, in this case the task context (see II-B), is applied for specifically addressing the UX factor "Feeling of Security".
- *Solution:* A solution must be described precisely and must not impose new problems. However, a solution describes only the core of the solution and the designer has the freedom to implement it in many ways. Other patterns might be needed to solve sub-problems; patterns relevant to the solution should be referenced, too. *An example for a CUX pattern solution in the application area car can be: "Support users to better estimate speed risks by using persuasive interfaces".*
- *Examples:* The example should show how the pattern has been used successfully in a system, i.e. refer to commonly known implementations of the pattern. Whether an example is commonly known or not can be difficult to determine, but this must be seen in relation

to the intended target audience for the patterns as well. Examples for real-life systems are preferably used, so that the validity of the pattern is enforced. *Examples for an applied CUX pattern are not available yet.*

Having the same structure for the whole pattern collection makes it easy for people to use them. As patterns represent possible solutions, which can be extended dynamically, they are never complete; when new solutions appear, they can be easily integrated into the given structure. This is in particular relevant for contextual interfaces, as the influence of context parameters on UX is a relatively new area of research.

V. Conclusion and Future Work

Within the CHI community, several experts commented on the pattern approach (see [38], most recently [39]). It was pointed out that patterns can facilitate the communication among all stakeholders and are more than just a sort of poetic form of guidelines. Pattern languages are intended to be meta languages used to generate project-specific languages that are grounded in the social and cultural particularities of a given design domain.

Moreover, it can be stated that patterns are important as they provide a common vocabulary [28], they are structured around the problems the designers face, and they provide solution statements [40]. These pattern characteristics can reduce time and effort for designing new projects considerably [16] and support a better communication among different stakeholders. In sum, capturing useful design solutions and generalizing them to address similar problems [12] is one of the big advantages of patterns, in part because the documentation and use of best practices improves the quality of design.

Within this paper, we especially build on patterns to introduce the "Contextual User Experience Patterns" (CUX patterns) approach. Thereby, we bring together the two relevant and increasingly important concepts of UX and context. UX is highly influenced and even changed by the context in which it occurs. So far, both concepts have been discussed a lot to various extent and in different dimensions, however, there is no guidance on how to design for a good UX for relevant application areas addressed in our research (car and factory). Thus, the introduced CUX patterns approach takes the discussion a step further by highlighting the potential of the patterns approach for describing knowledge on how to influence the users' experience in a positive way by taking context parameters during the interaction with a system into account.

Based on this theoretical argumentation, we describe the structure for CUX patterns, referring mainly to the application area car. This paper builds the starting point for our research within the recently established laboratory on contextual interfaces and will be further elaborated based on insights gained by ongoing and future empirical research.

Acknowledgment

The financial support by the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development is gratefully acknowledged (Christian Doppler Laboratory for "Contextual Interfaces").

References

[1] J. H. Westerink, M. Ouwerkerk, T. J. Overbeek, W. F. Pasveer, and B. d. Ruyter, *Probing Experience: From Assessment of User Emotions and Behaviour to Development of Products (Philips Research Book Series)*. Springer Publishing Company, Incorporated, 2008.

[2] A. Weiss, W. Reitberger, F. Pöhr, F. Förster, R. Buchner, and M. Tscheligi, ""Contextual interfaces" – Bridging the gap of fundamental research and industrial application," in *USAB2010: 6th Symposium of the WG HCI&UE of the Austrian Computer Society*. Springer, 2010, currently Submitted.

[3] M. Obrist, M. Tscheligi, B. de Ruyter, and A. Schmidt, "Contextual user experience: How to reflect it in interaction designs?" in *CHI EA'10: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2010, accepted for Publication.

[4] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, 1977.

[5] P. H. Kahn, N. G. Freier, T. Kanda, H. Ishiguro, J. H. Ruckert, R. L. Severson, and S. K. Kane, "Design patterns for sociality in human-robot interaction," in *HRI '08: Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*. New York, NY, USA: ACM, 2008, pp. 97–104.

[6] D. Wurhofer, M. Obrist, E. Beck, and M. Tscheligi, "Introducing a comprehensive quality criteria framework for validating patterns," in *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. IEEE Computer Society, 2009, pp. 242–247.

[7] J. McCarthy and P. Wright, "Technology as experience," *Interactions*, vol. 11, no. 5, pp. 42–43, 2004.

[8] M. Hassenzahl and N. Tractinsky, "User experience - a research agenda," *Behavior and Information Technology*, vol. 25, no. 2, pp. 91–97, 2006.

[9] E. L.-C. Law, V. Roto, M. Hassenzahl, A. P. Vermeeren, and J. Kort, "Understanding, scoping and defining user experience: a survey approach," in *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2009, pp. 719–728.

[10] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, pp. 4–7, 2001.

[11] S. Jumisko-Pyykkö and T. Vainio, "Framing the context of use for mobile hci," *International Journal of Mobile-Human-Computer-Interaction (IJMHCI)*, vol. 1, no. 2, 2010.

[12] A. Cooper, R. Reimann, and D. Cronin, *About Face 3: The Essentials of Interaction Design*. Indianapolis: Wiley, 2007.

[13] J. Coplien, “Organizational patterns: Beyond technology to people,” in *ICEIS (1)*, 2004, pp. IS–15.

[14] T. Coram and J. Lee, “Experiences - a pattern language for user interface design,” Website, 1998, available online at http://www.maplefish.com/todd/papers/experiences; retrieved at August 21st 2009.

[15] K. Kohler, S. Niebuhr, and M. Hassenzahl, “Stay on the ball! an interaction pattern approach to the engineering of motivation,” in *INTERACT (1)*, 2007, pp. 519–522.

[16] M. van Welie and G. van der Veer, “Pattern languages in interaction design,” in *Proceedings of IFIP INTERACT03: Human-Computer Interaction*. IFIP Technical Committee No 13 on Human-Computer Interaction, 2003, p. 527.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[18] E. Bayle, R. Bellamy, G. Casaday, T. Erickson, S. Fincher, B. Grinter, B. Gross, D. Lehder, H. Marmolin, B. Moore, C. Potts, G. Skousen, and J. Thomas, “Putting it all together: towards a pattern language for interaction design: A chi 97 workshop,” *SIGCHI Bull.*, vol. 30, no. 1, pp. 17–23, 1998.

[19] T. Erickson, “The Interaction Design Patterns Page,” Website, 2005, available online at http://www.visi.com/ snowfall/InteractionPatterns.html; retrieved at July 30th 2009.

[20] J. Tidwell, *Designing Interfaces : Patterns for Effective Interaction Design*. O’Reilly Media, Inc., 2005.

[21] M. Van Welie, “A Pattern Library for Interaction Design,” Website, 2005, available online at http://www.welie.com; retrieved at July 30th 2009.

[22] D. K. V. Duyne, J. Landay, and J. I. Hong, *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[23] A. Dearden and J. Finlay, “Pattern languages in hci: A critical review,” *Human-Computer Interaction*, vol. 1, pp. 49–102, 2006.

[24] J. Borchers, *A Pattern Approach to Interaction Design*. Chichester, England: John Wiley & Sons, 2001.

[25] D. Martin, M. Rouncefield, and I. Sommerville, “Applying patterns of cooperative interaction to work (re)design: e-government and planning,” in *CHI ’02: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2002, pp. 235–242.

[26] M. W. Steenson, “Feature problems before patterns: a different look at christopher alexander and pattern languages,” *interactions*, vol. 16, no. 2, pp. 20–23, 2009.

[27] J. Vlissides, “Patterns: The Top Ten Misconceptions,” Website, 1997, available online at http://www.research.ibm.com/designpatterns/pubs/top10misc.html; retrieved at July 30th 2009.

[28] T. Erickson, “Lingua francas for design: sacred places and pattern languages,” in *DIS ’00: Proceedings of the 3rd conference on Designing interactive systems*. New York, NY, USA: ACM, 2000, pp. 357–368.

[29] J. Zimmerman, “Designing for the self: making products that help people become the person they desire to be,” in *CHI ’09: Proceedings of the 27th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2009, pp. 395–404.

[30] K. McGee, “Patterns and computer game design innovation,” in *IE ’07: Proceedings of the 4th Australasian conference on Interactive entertainment*. Melbourne, Australia, Australia: RMIT University, 2007, pp. 1–8.

[31] E. S. Chung, J. I. Hong, J. Lin, M. K. Prabaker, J. A. Landay, and A. L. Liu, “Development and evaluation of emerging design patterns for ubiquitous computing,” in *DIS ’04: Proceedings of the 5th conference on Designing interactive systems*. New York, NY, USA: ACM, 2004, pp. 233–242.

[32] C. Crumlish and E. Malone, *Designing Social Interfaces*. O’Reilly, 2009.

[33] P. Kotzé, K. Renaud, and J. v. Biljon, “Don’t do this - pitfalls in using anti-patterns in teaching human-computer interaction principles,” *Comput. Educ.*, vol. 50, no. 3, pp. 979–1008, 2008.

[34] D. Martin, T. Rodden, M. Rouncefield, I. Sommerville, and S. Viller, “Finding patterns in the fieldwork,” in *ECSCW’01: Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work*. Norwell, MA, USA: Kluwer Academic Publishers, 2001, pp. 39–58.

[35] A. Crabtree, T. Hemmings, and T. Rodden, “Pattern-based support for interactive design in domestic settings,” in *DIS ’02: Proceedings of the 4th conference on Designing interactive systems*. New York, NY, USA: ACM, 2002, pp. 265–276.

[36] D. Wurhofer, M. Obrist, E. Beck, and M. Tscheligi, “Introducing a comprehensive quality criteria framework for validating patterns by means of a case study,” *Invited article for IARIA: International Journal on Advances in Software*, 2010, to Appear.

[37] M. Obrist, D. Wurhofer, E. Beck, A. Karahasanovic, and M. Tscheligi, “User Experience (UX) Patterns for Audio-Visual Networked Applications: Inspirations for Design,” in *Proceedings of NordiCHI 2010*. ACM, 2010.

[38] J. O. Borchers and J. C. Thomas, “Patterns: what’s in it for hci?” in *CHI ’01: CHI ’01 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2001, pp. 225–226.

[39] A. F. Blackwell and S. Fincher, “Pux: patterns of user experience,” *interactions*, vol. 17, no. 2, pp. 27–31, 2010.

[40] S. Fincher, J. Finlay, S. Greene, L. Jones, P. Matchen, J. Thomas, and P. J. Molina, “Perspectives on hci patterns: concepts and tools,” in *CHI ’03: CHI ’03 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2003, pp. 1044–1045.

A New Pattern Template to Support the Design of Security Architectures

Santiago Moral-García[1], Roberto Ortiz[2], Santiago Moral-Rubio[3], Belén Vela[1], Javier Garzás[1, 5],
Eduardo Fernández-Medina[4]

(1) Kybele Group. Dep. of Computer Languages and Systems II. University Rey Juan Carlos,
Madrid (Spain).
{santiago.moral, belen.vela, javier.garzas} @urjc.es
(2) S21secLabs. SOC. Group S21sec Gestión S.A, Madrid (Spain).
r.ortizpl@gmail.com
(3) Dep. Logical Security. BBVA, Madrid (Spain).
santiago.moral@grupobbva.es
(4) GSyA Research Group. Dep. of Information Technologies and Systems.
University of Castilla-La Mancha, Ciudad Real (Spain).
eduardo.fdezmedina@uclm.es
(5) Kybele Consulting, Madrid (Spain).
javier.garzas@kybeleconsulting.com

***Abstract*—The vast majority of current security patterns are oriented towards the production of security mechanisms, such as secure access systems or secure authentication systems. This type of patterns may be extremely useful for those security engineers who work on the production of this kind of mechanisms for large companies (Oracle, Microsoft, IBM, Google, Cisco, etc.), but they cannot be applied by a wide sector of security engineers who work in the development of security architectures. This is owing to the fact that these patterns do not consider aspects of the real complex system in which they will be installed. In order to complement security patterns and make them more applicable to security architecture design environments, in this paper we will propose a new description template of security patterns. The solution provided by this new template is oriented towards the architecture and technologies that should be used to design security architectures in real complex systems.**

Keywords: information security engineering; security architectures; security technologies; security patterns; real environments.

I. Introduction

Organizations currently require to guarantee availability, integrity and confidentiality of their assets [16]. In view of the fact that the realization of this task should consider the constant evolution of the organization's setting [27], we should specifically consider the variation between people, technologies, risks, processes, volumes of information, business strategies, etc. Therefore, there is a need to adapt the organization to all these changes in order to attain the objective of guaranteeing the fundamental security properties for its assets [20]. It is not easy for an organization to evaluate its level of risk and adapt itself to permanent changes. It is therefore vital for it to seek support from a security architecture [3] in order to mitigate the impact of these changes and thus minimize the risks associated with each of them.

The concept of security architecture can be defined as the practice of applying a structured, coordinated, rigorous method with the intention of discovering an organization's structure, bearing in mind human resources, business processes and technologies, i.e., all the elements that are involved in the organization to provide its systems with security and thus ensure the safety of its assets [19]. Security architectures are installed with the intention of minimizing the risks associated with the use of information technologies as well as optimizing an organization's business processes and strategies. If this objective is to be achieved, it is necessary to establish a set of technological infrastructure controls with which to identify the security mechanisms that are needed to define the system's security.

The security mechanisms used in security architectures are artifacts which have been designed to detect problems, prevent risks or make immediate corrections in order to avoid any undesirable events which may make security vulnerable [26].

After carrying out a systematic review of the literature related to security patterns, we have found out that the vast majority of patterns which are currently in use are focused on supporting the construction of new security mechanisms [9, 24, 28]. These patterns are a useful support for those engineers who work on developing security mechanisms which are the basic elements of an architecture [22, 7]. However, it is difficult to apply most of them to those work environments that are focused on the analysis and design of security architectures, since they do not consider the details of installing the solution in real complex systems [9, 28, 18]. We understand a real complex system to be all those elements that are involved in an organization, i.e., human resources, business processes and technologies.

We have therefore detected the need to discover structured solutions in the form of patterns, or the evolution of existing security patterns, to support information security engineers in the analysis and later design of security architectures which are used in an organization's real complex systems.

If security patterns are to be applicable to the sector of security engineers who design secure architectures in real systems, and confidentiality, integrity and availability of the organization's information assets are to be ensured, then it is necessary to resolve a series of lacks which have been detected. These solutions are shown as follows:

- Detailing the information assets, which the deployment of the pattern attempts to ensure, and the level of criticality of these assets.
- Detailing what an organization is protecting with the installation of the pattern.
- Including the deployment details in a real environment, bearing in mind the architecture and technologies that should be used to develop the solution in a satisfactory manner.
- Carrying out a qualitative analysis of the most important technological aspects with regard to the proposed solution (memory consumed, processing capacity, etc.).
- Bearing in mind different countries' rules and regulations with regard to the information assets that they wish to conserve. It may be that a solution which is legal in one country is not legal in another.

The lacks detected in current security patterns have led us to the belief that it is necessary to define a new description template of security patterns with which to resolve these limitations. This new template is characterized by the fact that it includes all the aspects which are necessary for a simple and reusable definition of security architectures. The definition of this template provides a step by step description of the architecture's design, and is linked to the necessary security requirements in relation to the criticality of the assets to be protected, known incidents, the systems involved in the solution, the necessary volumetric, and other variables associated with the environment such as the complexity of deployment, the use and maintenance of the solution, the regulations of the country in which the solution will be installed, and associated costs.

The remainder of this paper is organized as follows. Section II provides a description of the goodness of security patterns and shows related works in order to represent these patterns. Section III presents a new description template of security patterns. Section IV states our general conclusions with regard to the approach, and puts forward our future work.

II. Security Patterns

A security pattern describes a recurrent security problem which arises in a specific context, and provides a well tested generic scheme as a solution to that problem [12]. One of the main advantages of patterns is that they combine experience in the design of information system [10], thus making them more efficient. Patterns are a literary format with which to capture the knowledge and experience of security experts, resulting in a structured document in the form of a template to which the security experts' knowledge is transferred [21].

The first authors to propose security patterns were Yoder and Barcalow in 1997 [29]. The number of security patterns which have been published has increased considerably since then [22, 11, 30].

A great heterogeneity exists between the different descriptions found in each of the security patterns published [21, 15, 2, 13, 17]. This is because the authors who describe the security patterns that have been discovered have historically used different description templates to represent them. The most frequently used templates are those proposed by the Gang of Four [14], which have been adapted to describe security patterns, the template proposed by Buschmann et al. [4], the template proposed in the SERENITY project [23], and that proposed by Alexander [1]. Apart from these, other templates for the description of patterns have also been published, but their use is not massively extended yet. One example of these is that proposed in [25], in which the security patterns are represented as events calculus. Recent years have seen the proposal of other types of more specific security patterns, such as attack patterns [8] or misuse patterns [13].

As shown in [17], although the various authors who describe security patterns do not use a standardized description template, the majority of the description templates of these patterns have the following trio of elements in common: the context in which the pattern has been discovered; the security problem that is attempted to be resolved within the context put forward; and the forces that affect the solution. The solution is conditioned by the associated forces, and these are expressed through UML diagrams which model this solution [13].

In order to resolve the lacks detected in current security patterns and thus support information security engineers when analyzing and designing organizations' security architectures, we propose a new description template of security patterns. The template proposed below is intended to be an easy-to-use guideline which will allow both experts and non-experts in security to access a structured and methodical document with which to resolve security problems in the real complex systems of the organizations in which they work.

III. A New Description Template of Security Patterns

In this section, we shall set out the new description template of security patterns, explaining its characteristics and the contribution that it will make to the scientific community in the field of security. We shall then go on enumerating and detailing each of the description elements of the proposed template.

A security pattern focused on the development of security architectures describes a valid generic path that assists security engineers in making analysis and designing decisions when confronting the development of a secure architecture, which will resolve a real security deficiency in an information system. In order to obtain the maximum applicability within an organization, the proposed solution is oriented towards the architecture and technology that must be used in that organization to guarantee the security of the information assets associated with the deficiencies that we intend to resolve.

The new template will be described with the description elements from the description template proposed by Buschmann et al. [4] and the template proposed in the SERENITY project, used in [5], together with new description elements which are necessary to provide security experts and non-experts with a template to support the design of security architectures.

One of the main contributions of this approach is that the proposed solution provides security engineers with three complementary levels or *viewpoints:* platform independent level, platform specific level and product dependent level. This solution model manages to separate the implementation of the system's functionality specification over a platform in a specific technology. This allows differentiating the functionality that the system must satisfy and the technologies that could be implemented to develop the solution. Security engineers can also visualize the evolution of the solution from abstract models to real implementations in the complete system.

Figure 1 (below) shows a graphical representation of the solution levels.

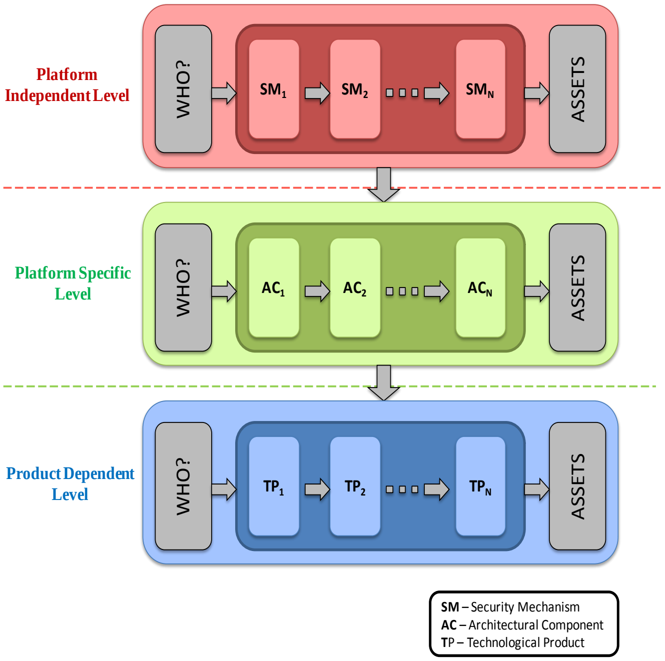


Figure 1. Abstraction levels of the solution.

As the figure above shows, all security systems must consider which information assets they intend to protect and who will have access to them.

We shall now provide a short description of each of the abstraction levels shown in Figure 1, and how the transformations through which to move from one level to the following should be carried out, illustrating the new elements needed to be incorporated or considered.

Platform Independent Level: this level provides a description of the security functionalities that the system should have, independently of its technological characteristics and implementation details. More specifically, a conceptual description of the security mechanisms that should be incorporated into the system is provided, along with the type of relationship that exists among them. The elements that should appear at this level are security patterns which are oriented towards the development of security mechanisms. A good guideline which can be used as a basis for discovering the type of patterns that are necessary is the guideline developed by Schumacher et al. in [22].

Platform Specific Level: the solution should be defined at this level, detailing the architecture or platform to which it will be applied. It is also necessary to set out how the necessary security mechanisms should be situated, through the presentation of an optimum security architecture with which to resolve the problem, independently of the technology used to protect the organization's systems. Given that security problems have repercussions on specific technological architectures, the same platform independent model can be instantiated N times, since it corresponds with different technological architectures. The security mechanisms described at the independent level become architectural components at this level.

Product Dependent Level: it is necessary to install the platform specific model into a specific architecture at this level, to implement it with technological products that are already available. Each of the architectural components can, therefore, be transformed into N technological products. The technological products must be valid products made by known manufacturers in the security industry. The final solution may vary significantly depending on the technologies used. This level should be independent of the information system's technological conditions. This view of the solution is very practical since it shows the user the different technologies that already exist on the market and that are oriented towards resolving the given problem.

This manner of structuring the solution provides a clear example of the steps that must be followed to implement the pattern, signifying that both experts and non-experts can understand the solution and know how to deploy it in a real system.

A further implicit property of this description template is its associated *decision path*. This element is of great assistance when selecting the most appropriate pattern with which to resolve a determined problem. The following five levels have been proposed in the decision path in order to classify the patterns that are associated with a discovered security deficiency:

1) *What is the state of the information, programs or configurations that need to be protected?* The possible states are the following:

a) Stored: These are found in a data base.

b) Transit: Through a transfer to another company or service. There is a movement of information.

c) Access: The information is being accessed.

2) *Who accesses the information that we wish to protect?* The people who can access the information are:

a) The organization's internal users.

b) External users or customers.

c) Computing staff during their work. This type of user is special since he can access data, applications and systems without using the security mechanisms which have been designed in the applications utilized by the final users.

3) *How is the information accessed?* or *What is the means of access?* In short, the information can be accessed in the following manners:

a) Directly: By accessing the data directly without any limitations on the use that is made of them.

b) Through an application: By applying business logic to the use, through which the information is shown.

4) *Where is the information accessed from?* It is basically accessed from two places:

a) Within the organization, i.e., all the technological spheres that are governed by the same security policies.

b) Outside the organization: where it is not possible to ensure the fulfillment of the same security policies that appear in the organization in which the assets are located.

5) *Who manages the means used to access the information that needs to be protected?*

a) The person responsible for security who will use the pattern and will be legally authorized to manage the systems' security.

b) Any other person who does not belong to the organization or does not have legal authorization to manage the system's security.

This *decision path* can be used to verify what type of problem, in general terms, will be resolved with the pattern discovered, i.e., two security patterns that respond identically to the same path resolve problems of the same nature, and could thus be alternatives to the same problem.

With regard to the elements described in the template, it is also necessary to emphasize that they do not describe the security vulnerabilities that may affect the information system in which the solution is installed. This is owing to the fact that new vulnerabilities frequently appear and the pattern must be constantly modified. We consider that the technologies themselves should be updated each time a new vulnerability is encountered, and that in this case it should be the manufacturer who updates them, or the security administrator who incorporates new rules into the security technologies used, if the impact of these vulnerabilities is to be minimized. This new template of security patterns therefore considers that vulnerabilities appear in all technologies on a permanent basis, and this concept forms a part of the pattern's considerations. The greater a technology's exposure to public networks, the higher its level of weakness. All security architectures will therefore be designed by bearing in mind that critical vulnerabilities repeatedly appear in all technologies.

The template proposed for the description of security patterns focused on the design of security architectures will be shown as follows. We must emphasize that this template is used to evolve existing security patterns, since it maintains the same base structure as their description, and it is only necessary to add the new elements that are proposed. The template that is proposed consists of the following elements:

A. Name

The pattern's name should represent the problem that it is attempting to resolve. This name must also be unique within the sphere of this type of patterns.

B. Context

The context provides a generic description of the setting, both at user level and system level, and includes the conditions under which the described pattern should be applied.

C. Problem

This describes the situation which has led to the necessity to apply a series of security mechanisms in order to obtain an optimum solution, and it basically details the reasons for the problem. It should also indicate the following questions:

- Which assets need to be protected? Information, programs and/or configurations.
- What are we protecting ourselves from? Information leaks, massive attacks, etc.
- Which security properties do we intend to conserve? Confidentiality, integrity, availability, auditability and/or non-repudiation.

D. Known incidents

It consists of a description of real cases of known security incidents, in relation to the problem posed that the implementation of the pattern intends to resolve. These incidents can be easily located on the Internet on specialized sites [6], which collect this type of events and specify when they occurred, how they occurred and what their impact was.

E. Decision Path

This element should describe all general levels of the state of the assets that need to be protected (previously described). This will make it possible to determine which pattern should be used to resolve a specific security problem. The objective of this descriptive element is to be able to develop a methodology based on security patterns, on the basis that the pattern's definition itself develops its own path in the decision tree.

F. Solution

This element describes the solution in accordance with the scenario and the problem being considered. This solution must be expressed at three different abstraction levels, as previously shown. It is first necessary to set out the solution for a platform independent level, showing the security mechanisms that must be used and the relationship that exists among them. This first level is then transformed into a second level, called platform specific level, which refers to the technological architecture proposed to resolve the given problem. The second level is finally transformed into a third level, called product dependent level, which shows a proposal for the technologies that can be used to implement the solution proposed by the described pattern. These technologies must be considered trustworthy by the Security Engineering sector.

Once these three levels have been developed, the solution should be complemented with a UML sequence diagram that is oriented towards the product dependent level, and that shows and describes in detail what the sequence of optimum processes to carry out the solution is.

G. Considerations

It is necessary to carry out a qualitative analysis of the solution in relation to the critical parameters found in the real complex system: a) storage; b) memory consumed; c) frequency with which the systems, technologies and applications are patched up; d) process capacity; e) complexity for final user; f) complexity for security/systems administrator; g) complexity of log management; h) broadband consumed; i) complexity for massive use of solution; j) cost of installing solution; and k) solution fulfillment guarantees. It is necessary to decide whether each of these aspects is qualitatively altered in a Null (0), Low (1), Medium (2) or High (3) manner when deploying the solution in a real information system.

These decisions will assist in the evaluation of whether or not the implementation of the solution is appropriate for the organization’s current situation. This is particularly true when considering the cost parameters and fulfillment conditions since excessive costs and an inability to ensure the fulfillment of the solution might be the main cause of any solution being rejected.

H. Rules and Regulations

If the adoption of a predefined solution in the form of a pattern in a real environment is desired, it is necessary to consider the regulations of the country in which the solution is intended to be installed, with regard to the information activities that need to be protected. We must also bear in mind the rules associated with these regulations which must be fulfilled by the proposed solution for it to be correct both juridical and legally. For example, Argentina does not permit the movement of information related to people who reside in that country and a solution which does not fulfill this regulation could not, therefore, be installed.

I. Benefits

A short description of a solution’s goodness with regard to the sphere and specific context in which the pattern is developed.

J. Consequences

This element describes the consequences of adopting a pattern as a solution in a real information system. An analysis of the risks that the organization runs if it does not adopt this solution must also be carried out. To do this, it is necessary to describe the following consequences:

- Negative consequences of adopting the solution.
- Consequences of not adopting the solution.

K. Alternatives

The majority of security deficiencies can be resolved in different ways, and this section should therefore describe other solutions that can be used to resolve the considered problem. These alternatives may differ from the pattern described at the technological level, at the architectural level or even in the security mechanisms used to guarantee the information assets that are at risk.

IV. CONCLUSION

In this paper we have presented a new description template of security patterns. To do this, we have provided a brief introduction to security patterns and their related works which put forward pattern description templates. We have then set out the reasons why security patterns focused on designing security architectures are necessary.

Existing security patterns are currently focused on supporting security engineers in the construction of security mechanisms. This type of patterns can rarely be applied by those security engineers who are dedicated to the analysis and later design of security architectures in real systems. This limited applicability results from the fact that current patterns: a) do not contemplate the impact of the systems involved in the solution; b) do not define the assets that must be protected; c) do not classify these assets according to their criticality; d) do not consider the restrictions involved in applying them in the different countries where we may wish to install the solution; e) do not consider the complexity of deployment, use and maintenance of the solution by the engineers in charge of them; f) do not define the reason why it is necessary to protect the assets; g) do not consider the impact of parameters on the system in which the solution will be installed; and h) do not put forward a real use case to provide both experts and non-experts in security with an example with which they can compare their problem. All of the aforementioned reasons led us to the belief that it was necessary to state a new description template of security patterns oriented towards resolving the need to obtain structured, valid and reusable solutions with which to support information security engineers in the analysis and design of security architectures in real complex systems.

We are currently working on the description of new security patterns focused on designing security architectures. We are also attempting to refine existing security patterns to make them applicable to the design of security architectures. Finally, we are defining a use methodology for this security patterns to allow both experts and non-experts in security to apply security to their systems in an easy, rapid and optimum manner.

ACKNOWLEDGEMENTS

This research has been carried out within the framework of the following projects: MODEL-CAOS (TIN2008-03582/TIN), ESPIA (TIN2007-67078) financed by the Spanish Ministry of Education and Science, QUASIMODO (PAC08-0157-0668), SISTEMAS (PII2I09-0150-3135) and SEGMENT (HITO-09-138) financed by the “Viceconsejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha” and the FEDER and BUSINESS (PET2008-0136) financed by the "Ministerio de Ciencia e Innovación (CDTI)” (Spain), and IDONEO (PAC08-0160-6141), financed by the “Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha”.

REFERENCES

[1] C. Alexander, S. Ishikawa, and M. Silverstein "A Pattern Language: Towns, Buildings, Constructions" Oxford University Press, 1977.

[2] Z. Anwar, W. Yurcik, R. E. Johnson, M. Hafiz, and R. H. Campbell "Multiple design patterns for voice over IP (VoIP) security" in Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International, 2006.

[3] A. Barth, C. Jackson, and C. Reis "The Security Architecture of the Chromium Browser" Technical Report 2008.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. "Pattern-oriented software architecture: A system of patterns" Wiley, 1996.

[5] A. Cuevas, P. El Khoury, L. Gomez, and A. Laube "Security Patterns for Capturing Encryption-Based Access Control to Sensor Data" in SECURWARE '08. Second International Conference on Emerging Security Information, Systems and Technologies, 2008, pp. 62-67.

[6] "DATALOSS db - Open Security Fundation", http://datalossdb.org/, 2010

[7] E. Fernandez "Security Patterns and Secure Systems Design" in Dependable Computing, 2007, pp. 233-234.

[8] E. Fernandez, J. Pelaez, and M. Larrondo-Petrie "Attack Patterns: A New Forensic and Design Tool" in Advances in Digital Forensics III, 2007, pp. 345-357.

[9] E. Fernandez, H. Washizaki, N. Yoshioka, A. Kubo, and Y. Fukazawa "Classifying Security Patterns" in Progress in WWW Research and Development, 2008, pp. 342-347.

[10] E. B. Fernández "Security patterns and secure systems design" ACM Southeast Regional Conference 2007.

[11] E. B. Fernandez and J. L. Ortega-Arjona "The Secure Pipes and Filters Pattern" in DEXA '09. 20th International Workshop on Database and Expert Systems Application, 2009, pp. 181-185.

[12] E. B. Fernandez, J. C. Pelaez, and M. M. Larrondo-Petrie "Security Patterns for Voice over IP Networks" in ICCGI 2007. International Multi-Conference on Computing in the Global Information Technology, 2007, pp. 33-33.

[13] E. B. Fernandez, N. Yoshioka, and H. Washizaki "Modeling Misuse Patterns" in ARES '09. International Conference on Availability, Reliability and Security, 2009, pp. 566-571.

[14] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides "Design Patterns: Elements of Reusable Object Oriented Software" Addison Wesley, 1995.

[15] J. Garzás and M. Piattini "Object Oriented Microarchitectural Design Knowledge" IEEE Software, pp. 28-33, 2005.

[16] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt "Security patterns repository, version 1.0" 2006.

[17] S. Moral-Garcia, R. Ortiz, B. Vela, J. Garzás, and E. Fernández-Medina "Patrones de Seguridad: ¿Homogéneos, validados y útiles?" in RECSI XI Tarragona, Spain, submit accepted.

[18] R. Ortiz, S. Moral-García, S. Moral-Rubio, B. Vela, J. Garzás, and E. Fernández-Medina "Applicability of Security Patterns" The 5th International Symposium on Information Security (IS'10 - OTM'10), 2010 - submit accepted.

[19] "OSA - Open Security Architecture", http://www.opensecurityarchitecture.org/cms/index.php, 2010

[20] D. G. Rosado, C. Gutiérrez, E. Fernández-Medina, and M. Piattini "Security patterns and requirements for internet-based applications" Internet Research: Electronic Networking Applications and Policy, 2006.

[21] M. Schumacher "B. Example Security Patterns and Annotations" in Security Engineering with Patterns, 2003, pp. 171-178.

[22] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad "Security Patterns: Integrating Security and Systems Engineering" Wiley, 2006.

[23] "Serenity Project - System Engineering for Security & Dependability", www.serenity-project.org, 2010

[24] M. Solinas, E. B. Fernandez, and L. Antonelli "Embedding Security Patterns into a Domain Model" in DEXA '09. 20th International Workshop on Database and Expert Systems Application, 2009, pp. 176-180.

[25] G. Spanoudakis, C. Kloukinas, and K. Androutsopoulos "Towards security monitoring patterns" in Proceedings of the 2007 ACM symposium on Applied computing Seoul, Korea: ACM, 2007.

[26] W. Stallings "Network security essentials: applications and standards", Prentice Hall, 2007.

[27] C. Steel, R. Nagappan, and R. Lai "Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management", Prentice Hall ed., 2005.

[28] H. Washizaki, E. B. Fernandez, K. Maruyama, A. Kubo, and N. Yoshioka "Improving the Classification of Security Patterns" in DEXA '09. 20th International Workshop on Database and Expert Systems Application, 2009, pp. 165-170.

[29] J. Yoder and J. Barcalow "Architectural Patterns for Enabling Application Security" Fourth Conference on Patterns Languages of Programs (PLoP'97), 1997.

[30] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen " An inventory of security patterns" Katholieke Universiteit Leuven, Department of Computer Science 2006.

A Pattern Collection for Privacy Enhancing Technology

Cornelia Graf[1], Peter Wolkerstorfer[1], Arjan Geven[1], Manfred Tscheligi[1,2]

(1) CURE
Center for Usability Research & Engineering
Modecenterstraße 17 / 2
1110 Vienna
+43.1.743 54 51
{last name}@cure.at

(2) ICT&S Center
University of Salzburg
Sigmund-Haffner Gasse 18
5020 Salzburg
+43.662.8044.4811
manfred.tscheligi@sbg.ac.at

Abstract— **Patterns are a useful approach to describe, organize and present solutions and best practices for design problems. Although much work can be found concerning either patterns or privacy, work focusing on patterns for Privacy Enhancing Technologies (PET) is very rare. This paper describes the development of User Interfaces Patterns for PET and the benefit of using them. We used different proven approaches and guidelines and merged them for creating efficient and useable patterns for PETs. Anyhow, some of our patterns were not tested much with end-users, so further tests will be necessary to prove worth of the patterns. Nevertheless we maintain that our patterns will support the development of PET UIs. In this paper, we provide a short overview of the whole pattern collection and present two patterns in detail.**

Keywords*- patterns; privacy; user interfaces desing; privacy enhancing technologies; pattern development; pattern approach.*

I. INTRODUCTION

Patterns provide solutions that have been successfully used for many years for specific problems. This paper describes the development of patterns for different User Interfaces (UI) for PETs and presents two of them more detailed. In current literature, patterns for PETs are very rare; approaches concerning patterns for privacy enhancing technologies are presented by [6], [10], [15], [16], [17] and [18].

The usage of PETs for privacy protection is a very important aspect for users' online life. Therefore, the lack of patterns concerning UIs for PETs is a big deficiency, for developers and designers as well as end-users. To help designers and programmers when dealing with the creation of UIs for privacy enhancing technologies we developed twelve patterns for PETs.

In the following Section, we will give an overview about the current research in the field of privacy patterns. Section III presents the development of our patterns approach and gives an overview of the PET patterns. In Section IV, we describe two of them in detail. In the last Section, we will discuss our approaches and give an overview about our future research.

II. RELATED WORK

In literature, much work can be found concerning either privacy or patterns but our literature research showed a lack of work concerning patterns for PETs.

Goldberg [3], [4] and Goldberg et al. [5] presented an overview about currently existing privacy enhancing technologies and gave an outlook of future PETs. Schumacher [17] presented two patterns, one deals with protection against cookies and the other one with pseudonymous mailing.

Schümmer [18] pointed out another privacy pattern approach, which covers the filtering of personal information in collaborative systems. These patterns address how to protect personal data from transmission to others and how to filter information received from others.

Other research concentrated on privacy protection and anonymity. Hafiz [6] presented a collection of privacy design patterns, which addresses anonymity solutions for various domains. Romanosky [15] developed three patterns, which describe how users can protect their privacy in web-based activities.

Another topic concerns patterns, which are related to privacy policies. Sadicoff [16] introduced a pattern especially for affording user awareness for privacy policies of network sites. The approach of Lobato [10] dealt with the development of user-friendly privacy policies.

In this paper, we present some patterns we created to support designers and developers when working on the development of UIs for PETs.

III. PET PATTERNS

Since the observing of privacy should be one main goal in users' online behavior, it is necessary to provide PETs for supporting them. Our current work deals with the development of different user interfaces for privacy enhancing technologies. As patterns provide useful and proven approaches for design problems we decided to use them for the development of our PET UIs. However, while we looked for patterns and proven approaches we concluded that only few patterns for privacy enhancing technologies are available and that they were not adaptive for our requirements. Hence, we decided to work not only on UIs for PETs but furthermore on development of patterns for assisting future PET developers.

The main requirement was to develop PET-UIs, which present the complex techniques of PETs in an understandable way to end-users. Another important requirement was to support users to protect their privacy in an active way. To achieve this we worked on solutions to make users aware of privacy related topics in web, like private data are requested, and so on. The online life of users can include many different use-cases of the Web; to cover all this use-cases with PETs it was necessary to develop several patterns for different tasks, i.e., support when creating a password, displaying if the privacy policy of a website matches with the preferred privacy settings and so on. Furthermore, we developed basic approaches for supporting users' privacy-behavior in the Web in privacy related way, i.e., information about visible private data in collaborative workspaces.

Our pattern approach should support the usage of best practice solutions for defined PET problems. Furthermore, they shall assist designers and developers as a guideline when creating UIs. This shall guarantee usable and consistent UIs in PET software. It should also enable capturing sharing and structuring of PET development knowledge inside the project team.

A. Pattern Development

Since we did not find feasible patterns for the needs of our UIs, we developed patterns by ourselves. A crucial factor when developing patterns is that each pattern must be consistent with the other patterns. The reason for this need for consistency is that a pattern describes not only a solution for a special problem but furthermore a solution for a special problem in a domain [8]. This means all patterns must veer toward the same purpose; - in our case, all patterns have to support the creation of user interfaces for PETs. Additionally, patterns are not allowed to contradict other patterns in the same domain. To create patterns, which complement one another in a meaningful way, we created our patterns during an iterative design process, which starts with the definition of the problem each pattern shall solve.

As a next step, we looked for approaches in literature and if guidelines on how to handle this problem already exist. We also integrated the knowledge, the experience and results which we gather in a predecessor project into the PET Patterns (e.g., [12]). This knowledge contains e.g., the observation that users try to get rid of intrusive privacy warnings by changing to more "generous" privacy preference settings.

Another point of interest were different guidelines of the European Union (EU) which describe privacy related concerns or approaches for supporting user perception of information ([2] e.g., Art. 25, Art. 7a, Art. 29 "Working Party") – these guidelines are commonly known as "European Data Protection Directive (DPD)".

Finally yet importantly, we looked for already proven approaches for our defined problems and adapted or merged them to create useable solutions. The most important approaches for the development of our patterns were:

- The multi-layered presentation approach, which is presented in Art. 29 Working Party [2]. This article recommends providing information in a multi-layered format under which each layer should offer individuals the information needed to understand their position and make decisions.
- Dynamic tooltips, which are based on the idea of motion design [7]. In motion design motion graphic are used for supporting the interactive system.
- An adaption of the "nutrition label for privacy" presented by Kelley [9]. Kelley adapted the nutrition label from the food domain and used it for displaying privacy policies to user.
- In addition, we also applied to the approach of Patrick [11], which lists four categories of human factors requirements for privacy interface design.

During the development, we combine guidelines and already proven approaches from the HCI. This grants already proven methods for the presented PET patterns.

For example, we used the multi-layered presentation approach and merged it with dynamic tooltips to inform users that private information is required. The "Dynamic Privacy Policy Display" which bases on this approach will be presented in Subsection IV.A.

Furthermore, we customized the "nutrition label for privacy" for a policy matching display. This display shall show users if and how much their preferred privacy settings matches with the policy of a website.

Experts first evaluated the outcomes of this merging through heuristic evaluation methods. Within this evaluation, they reviewed the outcomes based on classical usability principles. If this expert based analysis supports the design of the patterns we started with the next step – the end-user testing, otherwise we reworked the patterns. The goal of the end-user testing is, to look if the presented UI solutions of our patterns are understandable for end-users.

The end-user testing was done in several steps. If any problems were uncovered, we adjusted the pattern. This end-user testing is not completed yet, it will be continued in the further design of the patterns.

The goal of the patterns is to present complex technical PET mechanisms in an understandable way for users. The patterns will help designers and developers to create usable and supportive interfaces for PETs. For better usability of the patterns for designers and programmers we grouped the patterns, this shall provide a more efficient search for patterns for a special problem.

We classified our patterns through affinity diagramming [1] to grant designers and developers an efficient way to look through the patterns for solutions for PET concerning problems. The idea of grouping patterns can be also found at e.g., Welie [20] and Tidwell [19].

We didn't have categories in the beginning of the affinity diagramming; instead we analyzed the content and theme of each pattern and put related/similar patterns next to each other. In the end, we get three groups, which we named after the topic the patterns addresses.

Group 1: "PET Interaction" contains patterns, which are related to workflows and interaction paradigms.

Group 2: "PET patterns for privacy policies" includes patterns concerning the displaying of privacy policies.

Group 3: "PET Visualization" is related to the depiction of privacy information and to icons, which shall support a better visualization.

B. Patterns Structure

We based our patterns on the structure used by Welie [20] and extended it by a star rating, which illustrates the number of end-user testing we did with the pattern. Each Pattern consists of the following elements:

Title: The Name of the Pattern
Rating: A 0 to 5 star rating. It shows how much end-user tests were done.

- *Zero stars* mean that there weren't any end-user tests done with it.
- *One star* mean that low level HCI knowledge is included (in form of usability principles)
- *Two stars* mean that at least the user-feedback of two users was integrated.
- *Three stars* mean that more than two preliminary user evaluations have been done.
- *Four stars* means that a pattern is in a draft state and only misses a final iteration round.

  In Section IV, we will present two patterns with 4-stars rating.
- *Five stars* mean that much end-user testing was done and the results prove the content of the pattern; such patterns can be seen as final.

Problem: Summarizes and outlines the existing problem.
Solution: Brief description of the solution.
Use when: This group outlines the situation the pattern is best applied in.
How: More detailed insight into the solution
Why: Presents why the solution is needed and how to the user benefits from it.
Related Patterns: Refers to other patterns similar to the presented one.

From our point of view, this structure displays patterns in a plain way even for persons who never worked with patterns before and the appending of the star rating provides a first glance impression of how much end-user testing was done.

C. Overview of PET-Patterns

We developed twelve patterns for PET. Although all patterns are related to privacy, they can be clustered into different subgroups as described above.

In the following, we will give an overview of the three groups and the patterns related to them.

1) PET Intercation

The patterns in this group are related to workflows and interaction paradigms in PETs. This group has to offer elements, which can and shall be used in PETs to inform user about different topics concerning the approval of his data in the web. Patterns for following topics are attributed to PET Interactions:

a) Secure Passwords (****)

Secure passwords are a main concern for personal privacy protection.

This approach should help users to create and choose secure passwords by giving appropriate and dynamic feedback.

b) Informed Consent (****)

Users should fully understand what will happen if they release personal data in the web.

This UI solution should be used every time when the user needs to disclose personal data.

c) Privacy Aware Wording ( )

Users shall clearly understand the content and the terms of privacy policies. This approach should be used every time when a privacy policy will be displayed to a user.

d) Credential Selection (**)

This pattern should be used to make it easy for a user to select the appropriate credential and to inform him which data will the recipient have after the transmission.

2) PET Patterns for Privacy Policies

The patterns in this group are related to workflows and interaction paradigms in PETs. This group has to offer elements, which can and shall be used in PETs to inform user about different topics concerning the approval of his data in the web.

Following patterns can be found in this group:

a) Privacy Policy Display (***)

The goal of this display is to provide the user information about why what information by whom is requested.

It should be used whenever personal data are required from the user.

b) Dynamic Privacy Policy Display (****)

This pattern is presented in Section IV.

Display a tooltip to the user when his attention is required for privacy matters, e.g., person data are requested by a website.

c) Policy Matching Display (**)

Provide the user a possibility to compare each privacy policy with his preferred privacy settings. This approach should be used when a user contacts a service side or when the entry of personal data is required.

3) PET Visualization

PET Visualization offers suggestions on how to display privacy-related topics like "who sees which data" to the user.

The patterns in this group shall help designers to present this "who sees which data"-topic in an understandable way to users.

This group contains following patterns:

a) Privacy Icons (*)

Icons are able to speak for themselves, so icons are a great solution to aid written text. These icons should be used for PET software to support the user when he reads privacy information.

b) Icons for Privacy Policies ( )

Icons are able to speak for themselves, so icons are a great solution to support a user when he reads a privacy policy. These icons should be used together with privacy policies.

c) Privacy Awareness Panel in Collaborative Workspace (****)

This pattern is presented in Section IV.

The developed privacy panel shall users make aware of data (like IP or location) which are visible to others in a collaborative workspace. It should be used by collaborative workspace providers to help user protecting their privacy.

In the previous Section, we presented our patterns collection. The offered patterns cover various aspects of users' life. Through expert evaluation and previously done end-user tests, the patterns prove their worth for PET development. Furthermore, the multi-stage user tests showed that our solutions are understood by end-users. Therefore, we were able to fulfill our main requirement, an understandable presentation of the complex techniques of PETs to end-users. The second requirement was making users aware of privacy related topics. This requirement is integrated in different patterns, like "Secure Password" or "Policy Matching Display".

IV. Presentation of Selected Patterns

In the following Section, we will explain two selected patterns. First the "Dynamic Privacy Policy Display", which shall attract users' attention when needed, e.g., when private data are required; the second deals with an information panel for private data displaying in collaborative workspaces, it is called "Privacy Awareness Panel in Collaborative Workspaces".

We decided to present these two patterns because both are currently rated with four stars. This means that the patterns are still in draft state but only the final iteration round with end-users is missing. We think that these two patterns are good examples for the PET-Patterns we developed.

The presented patterns shall support user when login, reading and working on the web.

In accordance with Article 25 EU Directive 95/46/EC [2] individuals need to be informed about which of their data are processed, who is processing them and why. Therefore, it is necessary to inform users in an understandable way about what happened to their data and suggest possible consequences to them.

A. Dynamic Privacy Policy Display (****)

Problem

Users need to be well informed about possible consequences when releasing personal data upon certain actions such as login, registration, payments, etc. Art. 25 requires that data subjects are at least informed about what personal data are processed, by whom (i.e., the identity of the controller), and for what purposes [2].

Solution

The multi-layered presentation approach by the Article 29 Working Party [2] can be implemented by dynamical information "tooltips" informing the user about the nature of the data disclosed and possible consequences. The dynamic information need to be adapted to the context of the website it is used in. It should only include relevant security and privacy information and have a unique standard layout making it easy to recognize.

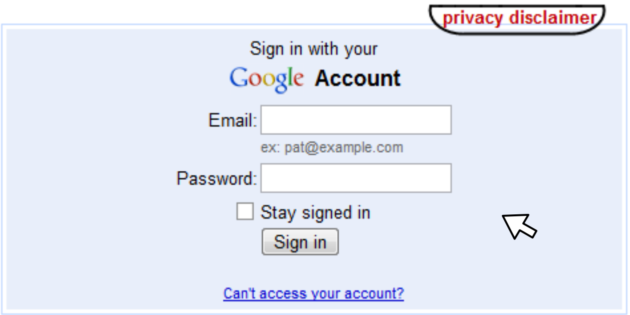


Figure 1: Prototype for Dynamic Display of Information

Figure 1 shows a prototype for dynamic display of information. When the mouse is moved nearby the interface, the privacy disclaimer in the top appears on the login-interface. This prototype was used for usability tests in Austria. To reduce the bias of the language it was designed in German.

Use when

Dynamic privacy policy displays can be applied to small interfaces (e.g., login) or when the credential selection contains information that needs the user's attention.

How

The information should be provided to the user where it is needed. Therefore the tooltip should appear on demand (i.e., need of information). This could be for example in a login dialog as soon as the user navigates the mouse into the concerning part of the interface (cf. Figure 1). The tooltip should then be made visible to the user and contain all necessary information for making an informed decision.

Why

Because of peripheral viewing, the user is able to recognize visual change (i.e., motion) even when on the border of the field of view. The user will recognize each visual change and might automatically connect it to danger. Hence, he will immediately notice the visual change and direct the attention to it. Using this approach, it is increasingly unlikely that the user might oversee the privacy indications.

Motion design is a known research area in the field of usability; thereby motion graphics are used as functional elements in interactive systems. According to Jacob [7], it decreases the cognitive load and creates user inputs – but only when applied correctly. Tooltips instead of pop-ups create a sense of seriousness (e.g., windows tooltips), whereas pop-ups are nowadays connected with error messages or unwanted advertisements. The physical connection between the tooltip and the Login dialog displays a certain attachment (i.e., that the tooltip is connected to the login dialog).

Research we did on the displaying of privacy preferences has shown that users recognize dynamic privacy policy display interfaces much better than static privacy policy displays. So 100% of our participants indicated to having recognized the dynamic box (visible in Figure 1), on the other hand, only 43% noticed a static display.

Related Patterns

- Privacy Policy Display

B. *Privacy Awareness Panel in Collaborative Workspaces (****)*

Problem

The problem with users' awareness for privacy in collaborative workspaces, e.g., forums or wikis, is twofold. First, the users can contribute under self-chosen nicknames instead of using their real names, which leads to a higher perceived anonymity of the users. However, providers of collaborative workspaces have more information about a user's real identity (e.g., IP address). Secondly, in collaborative workspaces, users disclose information – personal and non-personal – to an unknown audience. They have no idea how many and what kind of people can access their contributions. Both harm the person's privacy, even if the person is unaware of this.

Solution

In a so-called privacy-awareness panel, the audience is made transparent to the user, i.e., who can access his/her contribution (all internet users, registered users …). It is also pointed out that providers have additional information about the user. Hence, the privacy-awareness panel helps users to better understand their level of anonymity and private sphere within the collaborative workspace and based on this they can make better-informed decisions whether they want to disclose personal information in their contributions.

Use when

The approach should be used with every collaborative workspace.

How

First, it should be made clear to users which persons will be able to access their contributions. Second, users should know that providers get additional information about them for instance their IP addresses, browser versions, location information etc. and thus that they are not completely anonymous in the forum, wiki or other collaborative workspaces.

Further information about the Privacy Awareness Panel can be found in Poetzsch et al. [13].

Why

To allow users to make better informed decision whether they want to disclose personal data in their contributions to collaborative workspaces.

Related Patterns

- Privacy Options in Social Networks
- Selective Access Control in Forum Software
- Privacy Enhanced Group Scheduling

In the preceding Section, we presented two selected PET patterns from our collection.

The first presented pattern should be used every time when it is necessary to catch user's attention, e.g., when a user might oversee the privacy indications.

The second one, the privacy awareness panel, should be applied in every collaborative workspace to inform users which of their private data are visible to whom e.g., the provider.

Through implementation of these approaches, users will be better informed about what will happen to their private data and therefore be able to make informed decisions when dealing with their private data in web.

We advise that these approaches should be used every time to support users in making privacy aware decisions, although the patterns misses the final iteration round they prove their value during tests with end-users.

V. Conclusion and Future Work

In this paper, we have presented the development of UI patterns for privacy enhancing technologies. Furthermore, we have given an overview of the developed PET patterns and presented two of them in detail.

All the patterns in our collection have the same goal, helping designers and developers of PETs creating useable and understandable interfaces for end-users.

The merging of best practice solutions from HCI and different guidelines permitted us creating patterns for PETs with already proven solutions. Furthermore, the iterative development approves permanently improvements of the patterns. In a first step, experts evaluated each pattern and detected usability problems were remodeled.

Through end-user tests, we are able to identify user's problems of the different patterns and can therefore fix the uncovered usability problems in further version of the pattern. Furthermore, the patterns were rated with a star rating, which shows how much end-user testing has been done with each pattern. End-user testing has already been carried out with most of the pattern; only two have not yet been evaluated with end users. Four patterns are currently rated with four stars, so there is just the final iteration missing. However, for all patterns of our collection further user evaluations will be necessary.

Yet, results of end-user tests we made till now showed that we are able to fulfill our main requirement, the development of an understandable presentation of PETs for end-users. Anyhow, we maintain that the usage of our pattern collection for development of UIs for PETs will support users when dealing with private data in the web.

Future research will contain further usability testing of our presented patterns and based on the results of the tests a reworking and expansion of the UIs and patterns will be needed. The knowledge we gather from these evaluations will be continuously integrated into our existing patterns but it will also be necessary to create new patterns for new requirements.

The complete pattern collection can be found at [14].

ACKNOWLEDGMENT

Thanks to our PrimeLife partners for support, help and input while creating the patterns.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 216483.

REFERENCES

[1] H. Beyer. and K. Holtzblatt, “Contextual design: Defining customer-centered systems”. San Francisco, CA: Morgan Kaufmann, 1998.

[2] European Parliament, “Directive 95/46/EC of the European Parliament” , 1995. available at: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML; Last accessed : 2010-08-09

[3] I. Goldberg. “Privacy-enhancing Technologies for the Internet, II: Five Years Later”. In Workshop on Privacy Enhancing Technologies 2002, Lecture Notes in Computer Science 2482, pp. 1–12.

[4] I. Goldberg, “Privacy Enhancing Technologies for the Internet III: Ten Years Later”. In Acquisti A., Gritzalis S., Lambrinoudakis C., di Vimercati S. D. C. (eds.) Digital Privacy: Theory, Technologies, and Practices, Chapter 1. Auerbach, 2007.

[5] I. Goldberg, D. Wagner, and E. A. Brewer, “Privacy Enhancing Technologies for the Internet”. In COMPCON '97, pp. 103–109, February 1997.

[6] M. Hafiz, “A collection of privacy design patterns”. In Proc.of the 2006 Conference on Pattern Languages of Programs. PLoP '06. pp. 1-13.

[7] F. Jacob, “Ästhetik und UX: Das Potential von Serious Motion Graphics“, Xtopia 2008.

[8] D. Khazanchi, J.Murphy, and S. Petter , “Guidelines for evaluating patterns in the IS domain”. MWAIS 2008 Proc., Paper 24. http://aisel.aisnet.org/mwais2008/24; Last accessed : 2010-08-09

[9] P. G. Kelley, J. Bresee, L. F. Cranorand and R.W. Reeder, “A "nutrition label" for privacy”. In Proc.of the 5th Symposium on Usable Privacy and Security (SOUPS '09). pp. 1-12.

[10] L. L. Lobato and E. B. Fernandez, “Patterns to Support the Development of Privacy Policies”. First International Workshop on Organizational Security Aspects 2009 , pp.744-749

[11] A.S. Patrick and S. Kenny, “From Privacy Legislation to Interface Design: Implementing Information Privacy in Human-Computer Interaction”. Privacy Enhancing Technologies Workshop (PET 2003), Dresden/Germany, 2003, pp 107-124.

[12] J.S. Pettersson, S. Fischer-Hübner, N. Danielsson, J. Nilsson, M. Bergmann, S. Clauß, Th. Kriegelstein, and H. Krasemann, “Making PRIME usable”. In Proc. of the 2005 Symposium on Usable Privacy and Security (SOUPS '05), vol. 93. pp. 53-64.

[13] S. Pötzsch, P.Wolkerstorfer and C. Graf. Privacy-Awareness Information for Web Forums: Results from an Empirical Study. NordiCHI 2010, 16.–20. October 2010, Reykjavik.

[14] PrimeLife Project, http://www.primelife.eu; Last accessed : 2010-08-09

[15] S. Romanosky, A. Acquisti, J. Hong, L. F. Cranor, and B. Friedman, “Privacy patterns for online interactions”. In Proc.of the 2006 Conference on Pattern Languages of Programs (PLoP '06). pp. 1-9.

[16] M. Sadicoff , M.M. Larrando-Petrie, and E.B. Fernandez, “Privacy aware network-client pattern”. In Proc. of the 12th Conference on Patterns Language of Programming (PLoP’05), 2005. Available at: http://hillside.net/plop/2005/proceedings/PLoP2005_msadicoff0_0.pdf. Last accessed: 2010-08-09

[17] M. Schumacher, “Security patterns and security standards - with selected security patterns for anonymity and privacy”. In Proc. of the European Conference on Patterns Language of Programming (EuroPLoP'02), 2002. http://citeseer.ist.psu.edu/schumacher03security.html. Last accessed : 2010-08-09

[18] T. Schümmer, "The Public Privacy -- Patterns for Filtering Personal Information in Collaborative Systems," In Proc. of the Conference on Human Factors in Computing Systems (CHI ‘04) 2004. Available at: http://www.pi6.fernuni-hagen.de/publ/CHI2004.pdf; Last accessed: 2010-08-09

[19] J. Tidwell, “Designing interfaces.” - Sebastopol, Calif. [u.a.] : O'Reilly 2005.

[20] M.v. Welie, “Patterns in Interaction Design”. Available at: http://www.welie.com/patterns/index.php; Last accessed : 2010-08-09

Multiple Pattern Matching

Stephen Fulwider and Amar Mukherjee
College of Engineering and Computer Science
University of Central Florida
Orlando, FL USA
Email: $\{$*stephen,amar*$\}$*@cs.ucf.edu*

***Abstract*—In this paper, we consider certain generalizations of string matching problems. The multiple pattern matching problem is that of finding all occurrences of a set of patterns in a fixed text. This is a well studied problem, and several popular Unix utilities (grep, agrep, and nrgrep, to name a few) implement a host of algorithms to solve this problem. In this paper, we present both exact and approximate multiple pattern matching, using a uniform paradigm that generalizes the Shift-AND method of Baeza-Yates and Gonnet. Our algorithm is able to achieve better performance in certain searching scenarios when compared with the Unix utilities agrep and nrgrep, and should be considered being added to the grep family of searching algorithms.**

***Keywords*-string matching, approximate string matching, multiple approximate string matching.**

I. Introduction

In this paper, we consider certain generalizations of string matching problems. The most well known exact string matching algorithms are the Knuth-Morris-Pratt algorithm [1] and the Boyer-Moore algorithm [2]. A generalization to multiple pattern matching using keyword trees was proposed by Aho-Corasick [3]. Baeza-Yates and Gonnet [4] proposed a very fast and practical exact pattern matching algorithm in which the length of the pattern n does not exceed the length of a typical integer word (typically 32) in a computer so that bit shift and logical AND operations can be assumed to take a constant amount of time. The worst case time complexity is $O(m)$ where m is the length of the text and the algorithm takes $O(n)$ storage. They also proposed a generalization of the algorithm to handle arbitrary patterns with errors, called *approximate string matching*. Based on this and other independent ideas, Wu and Manber [5], [6] developed a whole suite of fast algorithms that can handle exact multiple patterns and approximate matches, patterns with unlimited wild card characters, patterns expressed by *regular expressions* and patterns with arbitrary cost for different edit operations (replacement, insertion and deletion operations). More recently, Külekci [7] developed a competitive exact multi-pattern matching algorithm for fixed length patterns by exploiting bit-parallelism. Additionally, many approximate string matching algorithms for multiple patterns have been developed which use a wide range of techniques and vary in performance based on the size and type of input [8], [9], [10]. Many different approximate string matching algorithms have been proposed in the literature including the class of algorithms called the *sequence alignment* algorithms using dynamic programming formulations. Interested readers are referred to the book by Dan Gusfield [11] and a recent research monograph by Adjeroh, Bell and Mukherjee [12].

An earlier version of this paper due to a regrettable oversight did not mention the following reference [5] which essentially presents the same approach we describe here. Our implementation has some advantages to their approach which will be described in Section III and IV.

We formulate the problem discussed in this paper as follows: The input to the problem is a set of patterns $\mathcal{P} = \{P_1, P_2, \ldots, P_s\}$ of size $n = \sum |P_i|$ and a text T of size m, both over a finite alphabet Σ. We will consider a generalization of the approximate string matching problem. The output is all substrings (a set of consecutive characters in T) and subsequences (a set of not necessarily consecutive characters in T) that are close to the set of patterns $\mathcal{P}$ under some similarity measures. We adopt the *edit distance* or the *Levenshtein* distance as the similarity measure. In particular, we want to find all patterns in T such that the number of edit operations do not exceed $\{k_1, k_2, \ldots, k_s\}$ from patterns $\{P_1, P_2, \ldots, P_s\}$, respectively. A string P_i, $(1 \leq i \leq s)$, is said to be at an edit distance k_i to a subsequence Q in T if we can transform P_i to Q with a sequence of k_i insertions of single characters in arbitrary places in P_i, deletions of single characters in P_i, or replacements of a character in P_i by a character in Q. If all edit operations are replacements, it is equivalent to the so-called *Hamming distance or mismatch* measure. The quantity k_i is typically a small positive integer $(0 \leq k_i \leq c)$ where c is a constant. If all k_i's are integer 0, the problem is reduced to the *exact multiple pattern* matching problem. Most approximate string matching algorithms reported in the literature assume k_i to be constant for all patterns.

We develop the exact and approximate algorithms and all their generalizations using a uniform paradigm that generalizes the Shift-AND method of Baeza-Yates and Gonnet. In this respect, it is the same approach used by Wu and Manber [5], but we allow k_i to be different for different patterns. We also use a simplified formulation of the problem using only six recurrence expressions and provide formal proofs of correctness of all the algorithms presented. We

Table I
COMPLETE M MATRIX FOR $\mathcal{P} = \{abc, axa, bc\}$ AND $T = \epsilon baxabcx$

	T	ϵ	b	a	x	a	b	c	x
P		0	1	2	3	4	5	6	7
a	1	0	0	1	0	1	0	0	0
b	2	0	0	0	0	0	1	0	0
c	3	0	0	0	0	0	0	1	0
a	4	0	0	1	0	1	0	0	0
x	5	0	0	0	1	0	0	0	0
a	6	0	0	0	0	1	0	0	0
b	7	0	1	0	0	0	1	0	0
c	8	0	0	0	0	0	0	1	0

Table II
COLUMN 5 TO COLUMN 6 OF M WITH INTERMEDIATE STEPS SHOWN

Column 5	Shift-Or(5)	$U(c)$	Column 6
0	1	0	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	0	0
0	0	0	0
1	1	0	0
0	1	1	1

show that such an approach leads to very fast and practical performance exceeding the performance of the best algorithms available in the Unix utilities. We use the Baeza-Yates paradigm of using a *binary* valued matrix M in which a few simple operations allow us to extend the algorithm to multiple pattern matching, both in the exact and approximate case.

Exact and approximate string matching algorithms find applications in database search, data mining, text processing and editing, lexical analysis of computer programs, data compression and cryptography. In recent years, string matching and sequence alignment algorithms have been used extensively in the study of comparative genomics, proteomics, disease identification, drug design and molecular evolution theory. The remainder of the paper is organized as follows: Section II lays out the exact multiple pattern matching algorithm, Section III gives its approximate matching counterpart, Section IV gives our experimental results, and we make our conclusions in Section V.

II. EXACT MULTIPLE PATTERN MATCHING

This section is included for the sake of completeness and to set up our notations. We also include a formal proof of correctness.

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_s\}$ be a set of patterns of size $n = \sum |P_i|$ and T be the text of size m preceded by a character ϵ which does not occur in any pattern. Define $P = P_1 P_2 \cdots P_s$ to be the concatenated patterns and M to be an $n \times (m+1)$ *binary* valued matrix with i running from 1 to n and j running from 0 to m. Entry $M(i,j)$ is 1 iff characters indexed by r through i of P exactly match the characters indexed by $i - r + 1$ of T ending at the character indexed by j, where r is the starting index of the pattern containing the character indexed by i.

The complete M matrix for $\mathcal{P} = \{abc, axa, bc\}$ and $T = \epsilon baxabcx$ is shown in Table I.

Notice that $M(5,3)$ is 1, indicating that ax of the second pattern ($r = 4$, $i = 5$) matches the last 2 characters of T ending at position 3, ax. Anywhere a 1 exists at the end of a pattern indicates an occurrence of that pattern being found in the text (for example, $M(6,4)$, $M(3,6)$, and $M(8,6)$). Hence, computing the s rows ending each pattern solves the exact multiple pattern matching problem.

The algorithm first constructs an n-length binary vector $U(x)$ for each character x of the alphabet. $U(x)$ is set to 1 for the positions in P where character x appears. From the example above, $U(a) = 10010100$.

The algorithm also constructs S and F, n-length binary vectors giving the start and end indices of all the patterns, respectively. Formally, $S(i)$ is 1 iff a pattern from $\mathcal{P}$ begins at i in P. Similarly, $F(i)$ is 1 iff a pattern from $\mathcal{P}$ ends at i in P. From the example above, $S = 10010010$ and $F = 00100101$.

Define $Shift\text{-}Or(j-1)$ as the vector derived by shifting the vector for *column* $j - 1$ down by one position and setting all *on* bits in S to 1. The previous bit in position n disappears. In other words, shift column $j-1$ down by 1 and perform a bitwise OR with S.

A. Algorithm

M is constructed by a very simple algorithm. Initialize column 0 to all 0. Column $j \geq 1$ is obtained by taking the bitwise AND of $Shift\text{-}Or(j-1)$ with the U vector for character $T(j)$. If we let $M(j)$ denote the jth column of M, then $M(j) = Shift\text{-}Or(j-1)$ AND $U(T(j))$. Table II shows one iteration of the algorithm from column 5 to column 6.

Once column j has been obtained, it can be checked for any found matches with the aid of the F vector. Let Z be the bitwise AND of $M(j)$ with F. All locations where Z is 1 indicate found matches, and can be extracted efficiently using the following bit trick. Given a binary number $X > 0$, in order to find the lowest order bit of X which is turned on, simply perform the bitwise AND of X with $\sim (X - 1)$, where $\sim$ is the bitwise complement. This will give the number 2^a, where a is the index of the lowest order bit turned on. Finally, this value can be subtracted from X and this process repeated until $X = 0$, meaning all matching locations have been found.

Notice that at any given time the algorithm only ever needs the previous column of M in memory when computing the next, so this algorithm is efficient in terms of memory. The number of bit operations is $\Theta(mn)$. However, when n is less than the size of a single computer word, each operation can be done very efficiently as single-word operations. Even when n is larger than a single computer word, each operation can be done as just a few single-word operations. Hence, for reasonably sized sets of patterns, this algorithm is efficient in both time and space regardless of the size of the text.

B. *Proof of Correctness of Exact Multiple Pattern Matching Algorithm*

We prove correctness of the algorithm described by induction on the columns of M. Column 0 of M is computed correctly, since it is set to all 0, and it corresponds to character ϵ which does not occur in any pattern in $\mathcal{P}$. Assume that all columns $j-1 < m$ are computed correctly. We will prove that column j is computed correctly.

Recall that $M(j) = \mathit{Shift}\text{-}\mathit{Or}(j-1)$ AND $U(T(j))$. By assumption, $M(j-1)$ is computed correctly. Thus $\mathit{Shift}\text{-}\mathit{Or}(j-1)$ is correct from column $j-1$. Namely, this vector will contain a 1 only in locations which match up until the $(j-1)-th$ character of T with some prefix of a pattern P_i *or* in locations which are the first character of some pattern. Note that because we only shift down by one, it is only possible for the last character of one pattern to interfere with the first character of another pattern. However, the first character of each pattern will always be set to 1 by the definition of $\mathit{Shift}\text{-}\mathit{Or}$, so no actual interference will take place.

Now this value is bitwise ANDed with $U(T(j))$, meaning only those locations which match with the current character will stay on. Thus, all previous matches that continue to match will stay 1, and any previous match that no longer matches will become 0, and any previous mismatch will stay 0. Therefore, $M(j)$ is correctly computed, and the algorithm correctly computes all columns of M.

III. Approximate Multiple Pattern Matching

We now address the l-edits problem of finding all approximate matches to a set $\mathcal{P}$ of patterns with at most l edit operations (replacements, insertions, or deletions). Recall that r is the starting index of the pattern containing character $P(i)$. Define $M^l(i,j)$ as a natural extension of $M(i,j)$, where $M^l(i,j)$ is 1 iff $P[r \dots i]$ can be converted to some suffix of $T[1 \dots j]$ with no more than l edit operations. The exact multiple pattern matching problem is a special case of this problem where $l = 0$.

The complete M matrices for $\mathcal{P} = \{abc, wxz, qrs\}$ with k values {2,2,2} and $T = \epsilon abdwxyzqt$ are shown in Table III.

Table III
Complete M matrices for $\mathcal{P} = \{abc, wxz, qrs\}$ with k values $\{2,2,2\}$ and $T = \epsilon abdwxyzqt$

		T	ϵ	a	b	d	w	x	y	z	q	t
	P		0	1	2	3	4	5	6	7	8	9
M^0	a	1	0	1	0	0	0	0	0	0	0	0
	b	2	0	0	1	0	0	0	0	0	0	0
	c	3	0	0	0	0	0	0	0	0	0	0
	w	4	0	0	0	0	1	0	0	0	0	0
	x	5	0	0	0	0	0	1	0	0	0	0
	z	6	0	0	0	0	0	0	0	0	0	0
	q	7	0	0	0	0	0	0	0	0	1	0
	r	8	0	0	0	0	0	0	0	0	0	0
	s	9	0	0	0	0	0	0	0	0	0	0

		T	ϵ	a	b	d	w	x	y	z	q	t
	P		0	1	2	3	4	5	6	7	8	9
M^1	a	1	1	1	1	1	1	1	1	1	1	1
	b	2	0	1	1	1	0	0	0	0	0	0
	c	3	0	0	1	1	0	0	0	0	0	0
	w	4	1	1	1	1	1	1	1	1	1	1
	x	5	0	0	0	0	1	1	1	0	0	0
	z	6	0	0	0	0	0	1	1	1	0	0
	q	7	1	1	1	1	1	1	1	1	1	1
	r	8	0	0	0	0	0	0	0	0	1	1
	s	9	0	0	0	0	0	0	0	0	0	0

		T	ϵ	a	b	d	w	x	y	z	q	t
	P		0	1	2	3	4	5	6	7	8	9
M^2	a	1	1	1	1	1	1	1	1	1	1	1
	b	2	1	1	1	1	1	1	1	1	1	1
	c	3	0	1	1	1	1	0	0	0	0	0
	w	4	1	1	1	1	1	1	1	1	1	1
	x	5	1	1	1	1	1	1	1	1	1	1
	z	6	0	0	0	0	1	1	1	1	1	0
	q	7	1	1	1	1	1	1	1	1	1	1
	r	8	1	1	1	1	1	1	1	1	1	1
	s	9	0	0	0	0	0	0	0	0	1	1

Notice that $M^1(5,4)$ is 1, indicating that wx of the second pattern ($r = 4$, $i = 5$) matches a suffix of T ending at position 4 with at most 1 edit operation, in this case the suffix being w and the edit operation being to delete $P(5) = x$ from P. $M^1(6,7)$ is 1, indicating that an occurrence of wxz is found in the text ending at position 7. Here, the suffix of T is $wxyz$, and the edit operation is to insert $T(6) = y$ into the pattern between x and z. Anywhere a 1 exists at the end of a pattern in M^l indicates an occurrence of that pattern being found in the text with at most l edit operations.

Formally, in order to compute $M^l(j)$ (for $l \geq 1$), the following six recurrences may be used, an improvement to the seven recurrences given in [5]. Simply take the bit-wise OR of the following expressions, where $A \Downarrow x$ denotes shifting column A down by x bits and discarding any bits which shift past bit n:

1) $M^{l-1}(j)$
2) $\mathit{Shift}\text{-}\mathit{Or}(M^l(j-1))$ AND $U(T(j))$
3) $M^{l-1}(j-1) \Downarrow 1$
4) $M^{l-1}(j-1)$
5) $M^{l-1}(j) \Downarrow 1$
6) $(S \Downarrow l)$ AND $U(T(j))$

Essentially this says that $P[r \dots i]$ will match a suffix of $T[1 \dots j]$, with at most l edit operations, iff at least one of the following conditions hold:

1) $P[r \ldots i]$ matches a suffix of $T[1 \ldots j]$, with at most $l-1$ edit operations
2) $P[r \ldots i-1]$ matches a suffix of $T[1 \ldots j-1]$, with at most l edit operations, and $P[i] = T[j]$
3) $P[r \ldots i-1]$ matches a suffix of $T[1 \ldots j-1]$, with at most $l-1$ edit operations, and $P[i]$ is replaced by $T[j]$
4) $P[r \ldots i]$ matches a suffix of $T[1 \ldots j-1]$, with at most $l-1$ edit operations, and $T[j]$ is inserted into P after character $P[i]$
5) $P[r \ldots i-1]$ matches a suffix of $T[1 \ldots j]$, with at most $l-1$ edit operations, and $P[i]$ is deleted from P
6) $P[r \ldots l]$ may be deleted and $P[i] = T[j]$

There is only one exception to this recurrence, which occurs when computing M^1. For M^1, you must also OR this result with the binary vector S to allow the first character to be replaced or deleted.

As in the exact case, we must take care to show that this recurrence does not cause any interference between patterns. The second, third, and fifth expressions all cause the current column to be shifted down by 1. But for all $l \geq 1$, $M^l(j)$ is 1 for all on positions in the S vector. Since shifting down by 1 can only cause the last character of some pattern to shift into the first character of the next pattern, then any 1 shifted into the first position of a pattern would have been set to 1 by the S vector. The other possible shift occurs in the sixth expression, where S is shifted down by l bits. The only way for this shift to overlap with other patterns is when $l \geq |P_i|$ for some i. In this case, the shift by l bits overlaps into bit $b = l - |P_i| + 1$ of P_{i+1}. But since $|P_i| \geq 1$, then $b \leq l$, and so this bit will be on for P_{i+1} since these b characters of P may freely be deleted. In fact, it is possible for the shift by l to overlap with patterns past P_{i+1}, but the same argument holds for why this overlap does not cause interference for all subsequent patterns. Thus, bit shifting does not cause interference between the patterns.

A. Algorithm

After establishing the recurrence for approximate pattern matching, the algorithm for computing all approximate matches from a set $\mathcal{P}$ of patterns to a text T follows quite naturally. The only substantial change from the exact matching case is that instead of a single F vector, we now have a set of F vectors. Let $K = max\{k_1, k_2, \ldots, k_s\}$. Then compute $F_0, F_1, \ldots, F_K$, where $F_j(i)$ is 1 iff pattern p from $\mathcal{P}$ ends at i in P *and* $j \leq k_p$.

There are several ways to organize computation of the M matrix, but it makes the most sense to compute each column j for all M^l before computing any column $j+1$ for any M^l. Of course, each column will be computed in increasing order of l from 0 to K. Each column of M^0 is computed using the recurrence given in the exact case, and each column of M^l for $l \geq 1$ is computed using the recurrence given for the approximate case.

Once column j is computed for all M^l, all matches can be obtained with the aid of the F vectors. Let Z_l be the bitwise AND of $M^l(j)$ with F_l, and Z be the bitwise OR of $Z_0, Z_1, \ldots, Z_K$. All locations where Z is 1 indicate found approximate matches, and can be extracted efficiently using the same bit trick described for the exact matching case. In this way we allow each pattern to have a unique number of edit operations allowed, which is a new contribution by our algorithm.

B. Proof of Correctness of Approximate Multiple Pattern Matching Algorithm

The proof follows by induction on M^l. M^0 is the exact case and so is correct from before. M^1 only allows one character to be edited. The first character can be a replacement by the special case for M^1. Otherwise, the first expression lets M^1 stay a match if M^0 was a match. The second expression allows 1-edits to continue to be 1-edits when $P[i] = T[j]$. The third expression allows character $P[i]$ to be replaced by character $T[j]$ after an exact match. The fourth expression allows a single character to be inserted into the pattern after an exact match. The fifth and sixth expressions allow a single character to be deleted from the pattern after an exact match or a single character to be deleted from the beginning of the pattern if $P[r+1] = T[j]$, respectively. This list handles all the ways a pattern can be a 1-edit from the text, and so M^1 is correctly computed.

Now assume that for some $l-1$, M^2 through M^{l-1} are computed correctly. We prove that M^l is computed correctly. When computing M^l, the possible events are that $(l-1)$-edits continue to be l-edits, l-edits match at the next pair of characters and may be kept as l-edits, or $P[i]$ is replaced, $T[j]$ is inserted after $P[i]$, or $P[i]$ is deleted. The first two cases are handled exactly by the first and second expressions of the recurrence, respectively. The next three cases are handled by the last four expressions, allowing for $(l-1)$-edits to continue to be l-edits by replacement, insertion of a character into a pattern, or deletion of a character (or set of initial characters) from a pattern, respectively. Since M^{l-1} is assumed to have been computed correctly, this correctly computes M^l.

Similar to the exact case, the number of bit operations is $\Theta(mnK)$, and the memory usage is $O(nK)$. However, when K is a small constant and n is the size of only a few computer words, this algorithm is very practical.

IV. Experimental Results

We have developed the algorithms described in this paper and written C code to implement the ideas presented. In this section we give some timing results, comparing our method with the current methods which are used in practice.

Our exact matching algorithm is competitive with fgrep (invoked using the grep -F command), the standard Unix utility for doing exact pattern matching on a set of patterns.

Table IV
TIMES COMPARING PATS TO FGREP

Text File	Text Size	Pattern File	Time	
English Text[1]	12MB	10 4–6 length words	PATS	.25s
			fgrep	.321s
English Text[2]	115MB	30 common English words	PATS	1.92s
			fgrep	2.83s
Random Text	100MB	30 common English words	PATS	2.1s
			fgrep	2.5s
Genome[3]	215MB	6 common motifs (short strings)	PATS	3.6s
			fgrep	3.8s

fgrep implements the Aho-Corasick algorithm, a linear time algorithm for searching for a set of patterns in a given text. Table IV shows our timing results comparing our exact matching algorithm, PATS, to fgrep. Our results show a modest improvement over fgrep in certain searching scenarios, especially when n is small and many patterns can be expressed as just a few computer words. Of course, when the number of patterns gets too large (and hence n starts to grow to more than just a few computer words), fgrep becomes faster and would be the preferred method.

Where our algorithm shows its true usefulness is when doing approximate pattern matching against a set of patterns. The Unix utility agrep written by Manber and Wu [6] is known to be one of the fastest for doing approximate pattern matching. Another popular utility for fast approximate pattern matching is nrgrep, written by Navarro[0] [13]. Our method does not always match the times of these methods when searching against a single pattern, but when looking for approximate matches against a set of patterns we can beat these current methods, sometimes by very large margins in appropriate search settings. This is largely due to the fact that existing tools for multiple pattern matching must be re-run for each approximate match, where our algorithm is able to do multiple pattern approximate matching in a single pass. Table V shows our timing results comparing our approximate matching algorithm, APATS, to agrep and nrgrep.

We also ran our algorithm against 2 sets of English texts, one small text[1] and one large text[2], varying k over all values from 0 to 8. The results are shown in Fig. 1 and Fig. 2.

Table V
TIMES COMPARING APATS TO AGREP AND NRGREP

Text File	Text Size	Pattern File	k	Time	
English Text[1]	12MB	30 common English words	1	APATS	.29s
				agrep	1.4s
				nrgrep	1.4s
English Text[2]	115MB	100 common English words	2	APATS	2s
				agrep	153s
				nrgrep	177s
Random Text	100MB	100 common English words	2	APATS	1.15s
				agrep	78s
				nrgrep	104s
Genome[4]	1.3GB	12 common motifs (short strings)	1	APATS	82s
				agrep	144s
				nrgrep	297s

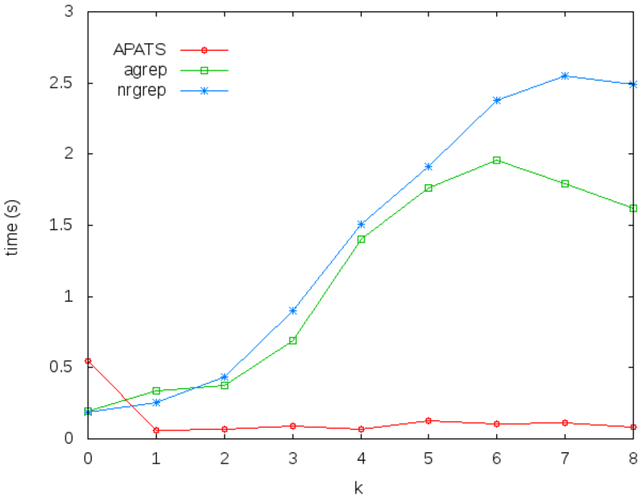


Figure 1. 12MB text searching for 30 common English words

As seen, when $k = 0$, agrep and nrgrep both exceed the performance of APATS. This is due to these algorithms being tailored to perform special exact pattern matching algorithms for the $k = 0$ case. However, for $k \geq 1$, APATS shows excellent performance, doing far better than both agrep and nrgrep for all values tested. Testing was limited to 8 due to limitations of the agrep software.

Our implementation can be downloaded at http://www.cs.ucf.edu/~stephen/pats-apats.

V. CONCLUSIONS

We have developed a very fast utility for exact and approximate pattern matching on a set of patterns. It is our hope that this algorithm would be added to the grep family of pattern matching algorithms and used in cases where it is expected to perform better than the current implementations, especially in cases where approximate pattern matching is

[0] We thank Dr. Navarro deeply for his personal communication and providing source code for our testing

[1] War and Peace from Project Gutenberg

[2] Selected works by Jane Austen, William Blake, Thornton W. Burgess, Sarah Cone Bryant, Lewis Carroll, G. K. Chesterton, Maria Edgeworth, King James Bible, Herman Melville, John Milton, William Shakespeare, and Walt Whitman from Project Gutenberg

[3] Chromosome 1 of Celera Genome from NCBI

[4] Chromosomes 1–6 of Human Genome from NCBI

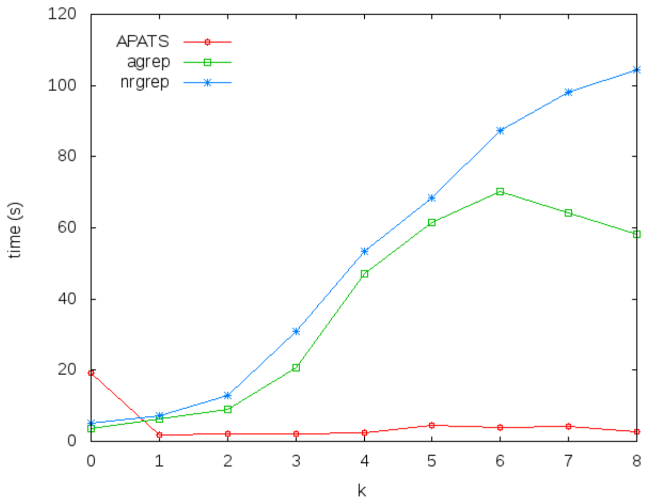


Figure 2. 115MB text searching for 20 English words of length 9–12

desired against a reasonably sized set of patterns. Another advantage of our algorithm is that it is very simple, both in concept and implementation. The run-times presented in this paper could no doubt be improved with a focus on optimizing implementation details. We have only implemented the algorithms exactly as they are presented.

References

[1] D. E. Knuth, Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[2] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[3] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[4] R. A. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," *SIGIR Forum*, vol. 23, no. SI, pp. 168–175, 1989.

[5] S. Wu and U. Manber, "Fast text searching: allowing errors," *Commun. ACM*, vol. 35, no. 10, pp. 83–91, 1992.

[6] ——, "Agrep - a fast approximate pattern-matching tool," in *In Proc. of USENIX Technical Conference*, 1992, pp. 153–162.

[7] M. Kulekci, "Tara: An algorithm for fast searching of multiple patterns on text files," *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on computer and information sciences*, pp. 1–6, 2007.

[8] R. Baeza-Yates and G. Navarro, "New and faster filters for multiple approximate string matching," *Random Struct. Algorithms*, vol. 20, no. 1, pp. 23–49, 2002.

[9] K. Fredriksson and G. Navarro, "Average-optimal single and multiple approximate string matching," *ACM Journal of Experimental Algorithmics*, vol. 9, 2004.

[10] H. Hyyrö, K. Fredriksson, and G. Navarro, "Increased bit-parallelism for approximate and multiple string matching," *ACM Journal of Experimental Algorithmics*, vol. 10, 2005.

[11] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology*. New York, NY, USA: Cambridge University Press, 1997.

[12] D. Adjeroh, T. Bell, and A. Mukherjee, *The Burrows-Wheeler Transform:: Data Compression, Suffix Arrays, and Pattern Matching*, 1st ed. Springer, July 2008.

[13] G. Navarro, "Nr-grep: A fast and flexible pattern matching tool," *Software Practice and Experience (SPE*, vol. 31, p. 2001, 2000.

Definition and Reuse of Analysis Patterns for Real-Time Applications

Hela Marouane, Saoussen Rekhis,
Rafik Bouaziz
Sfax University, BP 1088, 3018, Sfax, Tunisia
hela_marouane@yahoo.fr
{saoussen.rekhis, Raf.bouaziz}@fsegs.rnu.tn

Claude Duvallet, Bruno Sadeg
LITIS, UFR des Sciences et Techniques, BP 540,
76 058, Le Havre Cedex, France
{claude.duvallet, bruno.sadeg}@univ-lehavre.fr

***Abstract*— The analysis patterns improve the quality of products and the performance of development process. They have proven to be an effective means for capturing expert's knowledge and reducing the costs and the time of development. In this paper, we are interested in defining analysis patterns to model both the functional and non-functional requirements of Real-Time (RT) applications. The motivation behind the definition of these patterns is to facilitate the modeling of RT applications that must meet not only the accuracy of results, but also the time constraints related to the validity of data and the deadline of transactions. The proposed RT analysis patterns are illustrated through the modeling of two RT applications examples: the road traffic control and the medical control applications. These patterns are supported by a CASE toolset that both helps in RT analysis patterns representation and guides the patterns reuse.**

***Keywords*— Real-Time applications; analysis patterns; functional and non functional requirements.**

I. Introduction

Nowadays, Real-Time (RT) systems cover many sectors of activity: control of production lines, control of patients at home and medical assistance operations, control of road traffic, and so on. Generally, RT applications have common functionalities. Firstly, they acquire data from the environment by sensors. Then, they analyze the acquired data and provide results within the time constraints. Finally, they send orders to the environment via actuators. The design of these applications can be facilitated using reusable components that improve software quality and capture RT domain knowledge and design expertise.

There are different kinds of reusable components that can be applied in different levels of abstraction (analysis, design and implementation) such as software components, framework and patterns. Among these techniques, patterns have been the most widely used since they can be applied in different steps of the software development cycle. In order to benefit from the reuse at the first phase of development, several works [2] [3] [8] are interested in defining analysis patterns that provide facilities to model functional requirements of RT systems. The specification of functional requirements helps to understand the modularization of the structure of RT systems and to address the system's inputs, outputs, and their behavioral interrelationships. In addition, it is useful as a basis for RT systems design, test and documentation since the design of a developed system is evaluated from the functional point of view [7]. Nevertheless, requirements analysis must not exclude the modeling of non -functional aspects that define the general qualities of the intended product such as security, reliability, scalability, etc. That is, the concept of quality is also fundamental to software engineering, and the modeling of non-functional characteristics must be taken into consideration for early specification of restrictions and external constraints that RT systems must meet. Thereby, we interest in this paper to define RT analysis patterns that capture both functional and Non-Functional Requirements (NFRs) knowledge. In fact, to model the NFRs, UML profiles [15] [16] and the NFR Framework [1] expressed by Softgoals Interdependency Graph (SIG) can be used. The representation of NFRs with SIG makes their understanding easier. Moreover, the SIG is easy to adapt according to the systems evolution by adding softgoals and solutions through AND-decomposition and OR-decomposition. It is also easy to incorporate non-functional properties with functional requirements. For this reason, we adopt the NFR Framework [1] to model NFRs and we adopt UML use case diagram to represent the functional requirements of RT applications.

The remainder of this paper is organized as follows. Section 2 provides an overview of proposed analysis patterns that deal with the modeling of functional requirements of RT systems. Section 3 describes the definition of three analysis patterns to model the functionalities as well as the non-functional characteristics of RT applications. Section 4 illustrates the reuse of the proposed analysis patterns through the modeling of two examples of RT applications using our developed CASE toolset. Finally, we conclude in Section 5.

II. Related work

The term analysis pattern has been coined by Martin Fowler [4] for patterns which capture requirements in an application domain in order to allow reuse across applications. In this section, we present works on analysis patterns intended for the modeling of RT systems requirements. Among these works, there are the analysis patterns defined by Konard for the modeling of embedded systems [2] and "AMR" (Autonomous Mobile Robot) analysis pattern [3] used to model robot software.

The analysis patterns proposed by Konard [2] tend to have an inclination to focus primarily on either the structural or behavioral phase of object analysis. Therefore, they can be classified accordingly as structural object analysis patterns or behavioral object analysis patterns. In the following, we describe briefly two commonly used object analysis structural patterns, found in automotive embedded systems development: “Actuator-Sensor” and “User-Interface” patterns. (i) The analysis pattern “Actuator-Sensor” specifies basic types of sensors and actuators in embedded systems. It differentiates between four types of sensors and actuators: real, boolean, integer and complex. (ii) The analysis pattern “User-Interface” specifies the interaction between the user and the system through indicators (i.e., a type of actuators) and controls (i.e., a type of sensors).

The analysis patterns “AMR” (Autonomous Mobile Robot) [3] aim to develop and to facilitate the reuse of knowledge of robots software. Each pattern is described using both structural model and RT behavior model. The classes presented in structural model are classified accordingly to their RT behaviors as passive class, active class, event class, or implementation dependence class.

The examples of analysis patterns presented in this section focus on modeling the structural and behavioral aspects of embedded systems [2] and robot software [3], using UML class and sequence diagrams. The patterns proposed by Konard [2] represent also the functionalities of embedded systems using UML use case diagram. In fact, the representation of functional aspects shows clearly why RT systems are needed. But, the description of functionalities is not useful without the necessary non-functional characteristics such as dependability, reliability and security [9]. We must take into account the definition of these characteristics for the improvement of RT system quality and longevity [9]. Also, the quality of software system can only be achieved by considering non functional requirements as early as possible. If the NFRs are not considered at the early stage of the software development process (i.e., analysis phase), it may be difficult and expensive to address them in final product and it can lead the failure of the development.

III. RT ANALYSIS PATTERNS DEFINITION

In this section, we define three analysis patterns to model both the functional and non functional requirements of RT applications. The first pattern aims to model the data acquisition from the environment using sensors. The second pattern allows to model the control of data acquired from environment. While the third pattern deals with the representation of corrective actions when a violation is found.

In order to simplify the understanding of proposed analysis patterns, we describe these patterns using the following four elements: name, context, intention and solution. The solution shows how to model the RT applications requirements using UML and Softgoals Interdependency Graph [1]. The NFRs represented by Softgoals are associated with four use case diagram elements: actor, use case, actor-use case association and the system boundary. For example, in Figure 1, Dependability NFR is represented as Dependability [Transmission System] softgoal (denoted by a light cloud icon) that is related to the association between ‘Sensor’ actor and ‘Receive data from the environment’ use case. NFR softgoals are named using Type[Topic] nomenclature where Type represents a specific NFR concept e.g. Dependability, Security and Topic represents the context of the NFR [10]. The leaf-nodes of the graph represent alternative solutions for the operationalization of the NFR softgoals. They are denoted by dark clouds in the graph. Their corresponding degrees of contribution, indicating how well these solutions achieve NFR softgoals, are represented by the following signs: (MAKE (++), HELP (+), HURT (-), or BREAK (--)) [10].

A. The “Data Acquisition” analysis pattern

- **Name:** “Data Acquisition”.
- **Context:** this pattern is applicable in all RT applications which manipulate important volumes of data during the data acquisition phase.
- **Intention:** this pattern describes the functions as well as the quality that RT systems must have when acquiring data from the environment.
- **Solution:** Figure 1 shows the “Data Acquisition” analysis pattern that describes the interaction between the system and sensors. The sensors i.e., radar, camera, acquire data from the environment. Then these data are stored in RT databases. In distributed RT applications, data are transmitted to the databases of different sites with minimum time and cost of communication.

This pattern describes also the Softgoals Interdependency Graph for achieving NFR dependability. This graph represents a comprehensive set of software quality attributes related to dependability of data transmission system. This latter must be operable and able to perform its required function at any instant during its specified operating time. The dependability can be further achieved by ensuring data security and data transmission reliability.

- Security [Data]: the transmitted data from sensors to RT system must be secured against unauthorized accesses. A RT system may be useless if it does not satisfy security property. The NFR security is composed of availability, integrity and confidentiality. Availability means guarding against the interruption of service [11]. It can be achieved by replicating data. Integrity means guarding against unauthorized updates or other tampering. Confidentiality means guarding against unauthorized disclosure, i.e., release of relevant data [11]. To ensure integrity property, the transmitted data must be complete and accurate. The accuracy is the ability of a measure (e.g. speed, altitude, temperature, etc.) to match the actual value of the quantity being measured.

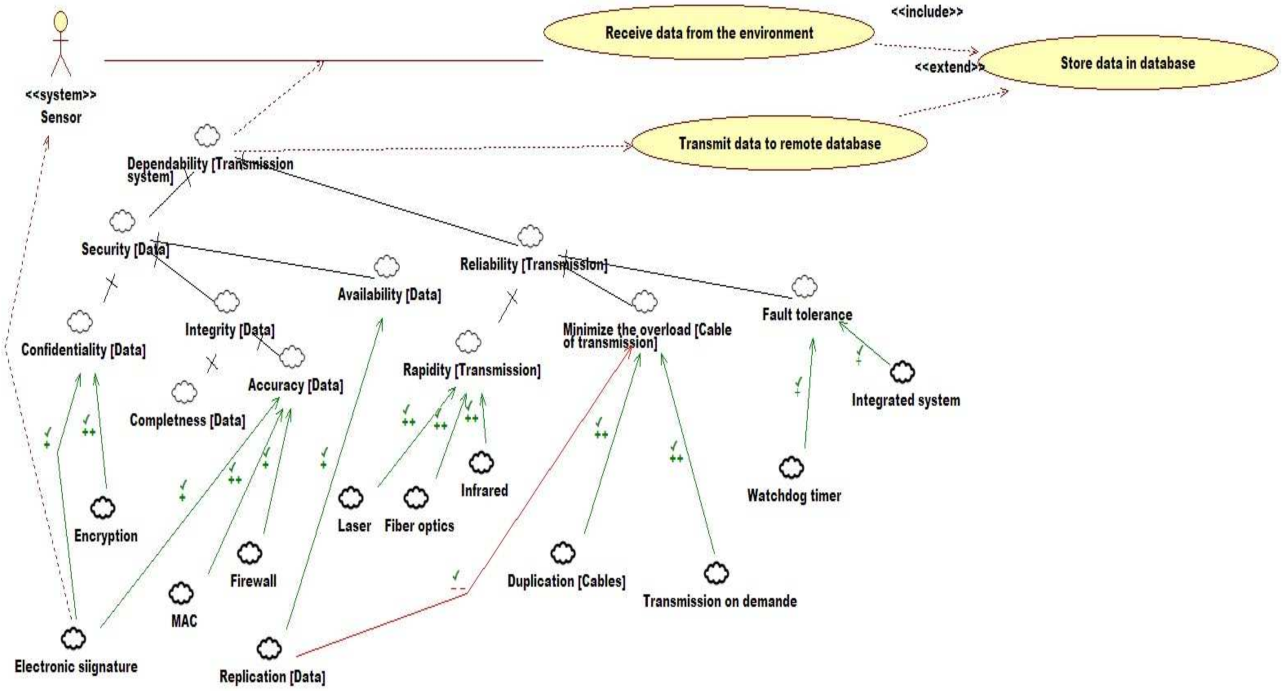


Figure 1. “Data Acquisition” analysis pattern

This constraint is ensured by the use of firewall that can block unauthorized accesses, by electronic signature based on encryption technology using public key and private key to authenticate the sender, or by using Message Authentication Code (MAC). The electronic signature can be also used to achieve confidentiality.

- Reliability [Transmission]: the transmission of data must be accomplished with minimum errors and time. In fact, the rapidity of data transmission is crucial in the RT applications since they must fulfill temporal constraints. It aims to ensure the respect of the validity of acquired data. If this quality is not satisfied, the data will not be fresh and therefore, lose their validity. Lasers, optical fiber and infrared are solutions to accelerate sensor data transmission. Besides, the degraded transmission mode which still corresponds to the specification is acceptable in RT system. This means that the imprecision of transmitted data is tolerable provided that they are received in time and they do not exceed the tolerable deviation. Fault-tolerance can be also applied by means of: (i) the watchdog timer, which is a device that triggers a system reset if the system does not respond due to some fault condition, (ii) and integrated system, which corresponds to a runtime equipment or software support for both real-time and fault tolerance.

B. The analysis pattern “Control of the environment”

• **Name:** “Control of the environment”.

• **Context:** this pattern is applicable in all RT applications which manipulate important volumes of data during the control phase.

• **Intention:** this pattern is used to model how control system monitors the acquired data and detects failures in RT applications.

• **Solution:** Figure 2 describes the shared and the varying functions of control system as well as their non functional characteristics. The variability is expressed through the generalisation and <<extend>> relationships. The generalisation relationship specifies different problems that can be detected in RT systems. Whereas the extension relationship specifies that the control of the environment functionality can be extended by the detection of errors related to data or actions. In fact, the control system can detect the non freshness of data if a measure’s value is used out of time interval during which it is considered valid. The control system can also detect an anomaly if boundary constraints are not fulfilled, i.e., a measure’s value is not between the minimum value and the maximum value defined by the user. For example, in freeway traffic management system, if the road segment density exceeds the limit, then the control system reports an anomaly. In addition, the control system can detect that the Quality of Data is not fulfilled (QoD) [12] if the maximum data error, defined to allow imprecise RT data, is exceeded. Note that the maximum data error is the upper bound of the difference between the value stored in the database and the new value acquired from sensor. Besides, the control system can detect an error if the deadline of an action is missed.

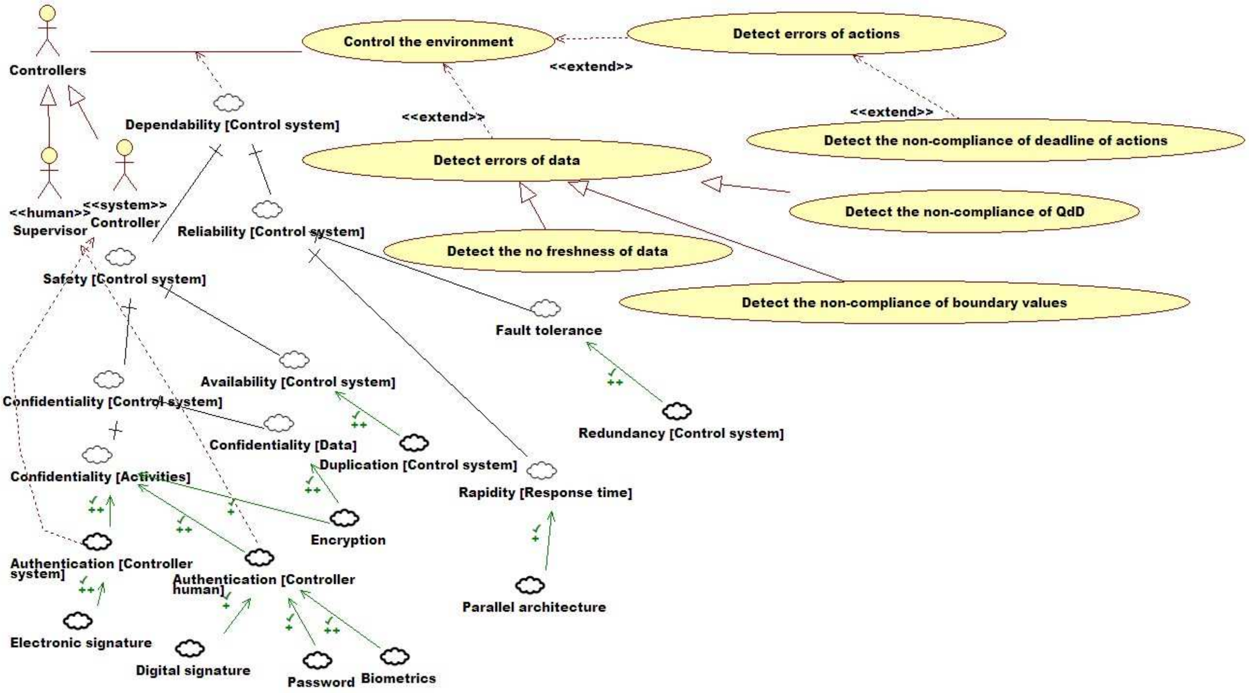

Controllers
Control the environment
Detect errors of actions
<<extend>>
<<extend>>
<<extend>>
Dependability [Control system]
Detect the non-compliance of deadline of actions
Detect errors of data
<<human>>
Supervisor
<<system>>
Controller
Reliability [Control system]
Detect the non-compliance of QdD
Detect the no freshness of data
Safety [Control system]
Detect the non-compliance of boundary values
Fault tolerance
Availability [Control system]
Confidentiality [Control system]
Redundancy [Control system]
Confidentiality [Data]
Duplication [Control system]
Confidentiality [Activities]
Rapidity [Response time]
Encryption
Authentication [Controller system]
Authentication [Controller human]
Parallel architecture
Electronic signature
Digital signature
Password
Biometrics
++
+

Figure 2. "Control of the environment" analysis pattern

When monitoring the environment, the control system must operate correctly. Thereby, it must fulfill the dependability property that is defined as the ability of the system to deliver specified services to the end users [13]. The dependability includes safety and reliability characteristics.

- Safety [Control system]: the control system must be protected against unauthorized accesses and must be available at all times. Availability means that the control system must be accessible at any time. For example, in case of failure, it must have a backup system. Confidentiality means that the control should be made by authorized members. This authorization is assigned by an authentication procedure. If the controller is a person, the authentication is ensured by password, biometrics or digital signature methods. Biometric can be implemented either by finger print verification or by voice or by face recognition. Digital signature is based on the technique of encryption with public key that is used to verify the signature and private key that is used to sign. Nevertheless, if the controller is a system, the authentication is ensured by electronic signature. Thus, only authorized controllers can exchange data and messages.

- Reliability [Control system]: The reliability is the ability of a system to consistently perform its intended or required function in time. Thereby, the control system must report the identified errors in a minimum time. In order to ensure the rapidity of response time, using parallel architectures is an effective solution. Besides, the reliability may be achieved with fault prevention by using different versions for the same processing.

C. The "Sending orders" analysis pattern

• **Name:** "Sending orders".

• **Context:** this pattern is used in RT applications when the control system reports an error that is occurred in the environment.

• **Intention:** this pattern aims to model the different recovery actions that are activated by actuators.

• **Solution:** Figure 3 illustrates the "Sending orders" pattern that describes the functional and non-functional requirements of actuator system. It allows to express variability since it shows different kinds of actions that can be triggered when the control system reports an error to the actuators. Indeed, actuators can trigger actions to achieve. It can also report alert messages such as voice messages, visual messages or alarms.

The non functional characteristics of actuator system are the same than those explained in the "control of environment" pattern.

IV. RT ANALYSIS PATTERNS REUSE

We have developed a toolset, called *AP-RT* (Analysis Pattern for RT applications), that deals with patterns representation and guides analysis patterns reuse. AP-RT allows the user to apply the proposed patterns in order to

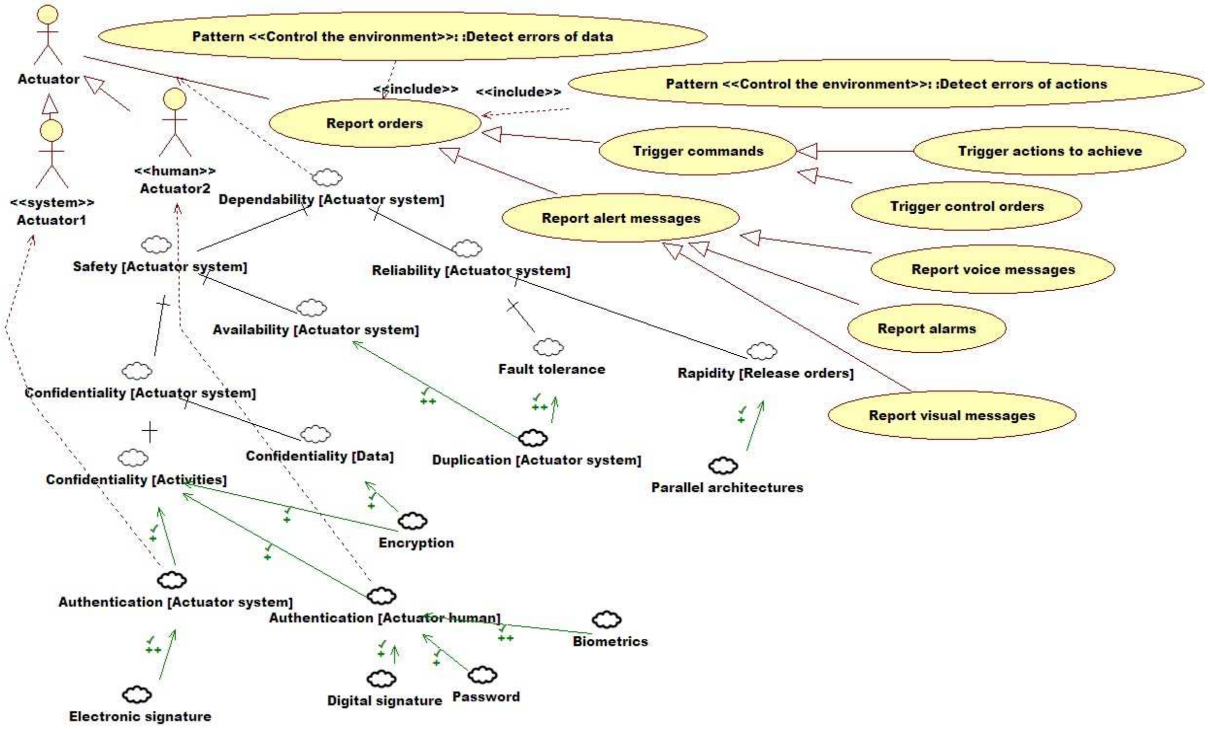

Actuator
<<system>>
Actuator1
<<human>>
Actuator2
Pattern <<Control the environment>>: :Detect errors of data
<<include>>
<<include>>
Pattern <<Control the environment>>: :Detect errors of actions
Report orders
Trigger commands
Trigger actions to achieve
Trigger control orders
Report alert messages
Report voice messages
Report alarms
Report visual messages
Dependability [Actuator system]
Safety [Actuator system]
Reliability [Actuator system]
Availability [Actuator system]
Fault tolerance
Rapidity [Release orders]
Confidentiality [Actuator system]
Confidentiality [Data]
Duplication [Actuator system]
Parallel architectures
Confidentiality [Activities]
Encryption
Authentication [Actuator system]
Authentication [Actuator human]
Biometrics
Electronic signature
Digital signature
Password
++

Figure 3. “Sending orders” analysis pattern

facilitate the modeling of functional and non-functional requirements of specific RT applications. As illustrated in Figure 4, the pattern can be created within AP-RT tool by means of the menu bar shown in the left of the figure. In addition to presenting patterns, AP-RT assists the designer in instantiating an analysis pattern selected and adapted via the menu bar shown in the top of Figure 4. After creating an initial application model that instantiates a pattern, the designer can continue developing the application model by adding, updating and removing various model elements using the AP-RT tool. Note that the tool does not allow the remove of pattern elements that are shared between all RT applications (i.e., pattern fundamental elements).

In the following, we describe the instantiation of defined analysis patterns using AP-RT tool. We illustrate the reuse of “Data Acquisition” pattern through the modeling of a road traffic control system. Also, we reuse the “Data Acquisition” and “Control of the environment” patterns in order to model a medical control application.

A. Example of a road traffic control system modeling

The road traffic management systems have become an important task intended to improve safety and provide a better level of service to motorists. We focus on modeling the acquisition data subsystem of a road traffic control application [6] and we explain how this design issue can be facilitated by the reuse of the “data acquisition” pattern. Current road state is obtained from the essential sources: radars, inductance loop detectors and supervision cameras. This system uses radars to measure vehicle velocity and to acquire vehicle crossing time of red light. It uses also inductance loops to measure traffic density (i.e., number of vehicles in a road segment). The supervision cameras are used to supplement and to confirm the data received through the vehicle detector stations and to provide information on local conditions which affect the traffic flow. The road traffic management system analyses the acquired data by the Central Computer System and informs drivers in real time about the state of circulation using variable message signs.

In order to ensure the security of data transmission, the road traffic control system maintains the confidentiality and the data integrity characteristics using electronic signature. In addition, the system maintains the rapidity through data compressing and laser technology. The data compressing is also used to minimize the overloaded transmission.

Figure 4 shows how to adapt the “Data Acquisition” pattern to model this system. The “Sensor” actor is instantiated by “Radar”, “Inductance loop” and “Camera”. Then, the “Receive data from the environment” use case is instantiated by the corresponding road traffic system functions which are: (i) “Receive vehicles speeds and crossing times” use case associated to the “Radar” actor, (ii) “Receive road segment density” use case associated to “Inductance loop” actor and (iii) “Receive image of vehicles” use case associated to the ”Camera” actor.

The operationalizing softgoals, associated to road traffic control system, are: “Electronic signature”, “Laser” and “Compression data”. These desirable leaf-node solutions are labelled with (√) sign. Whereas, the proprieties that do not have Operationalizing Softgoals are labelled with (☓) sign.

B. Example of a medical telesurveillance system modeling

Telemedicine for patient in residence, called “Televigilance”, concerns elderly persons, people with cardiac pathologies and persons in convalescence after hospitalisation, all needing a close medical supervision. The MEDIVILLE system [5] for telesurveillance of patients at home allows a more reactive medicalisation remotely released by urgency units (diagnosis, intervention).

This system is composed of three main components: (1) a terminal placed on the patient, continuously recording his physiological data, (2) an in-door reception base-station, processing physiological signals to detect emergency situation and create an alarm, which is retransmitted to the (3) third component corresponding to a remote medical monitoring server hosted in the televigilance centre exploiting all these data to decide any intervention. The patient’s terminal is coupled to actimetry and pulse sensors, indicating respectively the attitude of the patient (vertical/horizontal positions, activity) and his heart rate (pulse measurement). The base station continuously receives the emission signals from the patient’s terminal through a VHF radio link. On the other side the base station is connected to the remote server of the Surveillance centre through an IP channel using a VPN (Virtual Private Network) protocol.

During normal operation, the local home system is solicited at regular intervals (every 30 seconds) by the remote server in order to transmit the totality of recent sensors data. In an alarm case, the local system communicates with the central server and transmits, simultaneously to the alarm, the latest available data.

The MEDIVILLE system ensures a good quality of service. Firstly, an original noise reduction algorithm implemented in the microcontrollers aims to reduce the variations of pulse measurement and then to improve data accuracy. Secondly, the access to stored information is allowed only for authorised users: patient agent, practitioner agent and server manager. Thirdly, the importance and complexity of the functions taken over by this server strongly require a reliable system. The system contains a sufficient capacity and redundancy in order to cater for these conditions. Finally, the WS-DSAC *(Web Servers – Differentiated Services Admission Control)* [14] mechanism is used to accelerate response time. This mechanism is based on the balancing of imposed load among a certain number of computers to improve performance and on the use of admission control mechanisms to allow differentiated allocation of resources for specific service classes [14].

Reuse of the “Data acquisition” pattern: Figure 5 shows the reuse of the pattern “Data acquisition” to model the medical telesurveillance system.

The “Actimetry_Sensor” and “Pulse_Sensor” represent the instances of “Sensor” actor. These sensors are associated respectively to the following use cases: “Receive the attitude of the patient” and “Receive the pulse measurement”.

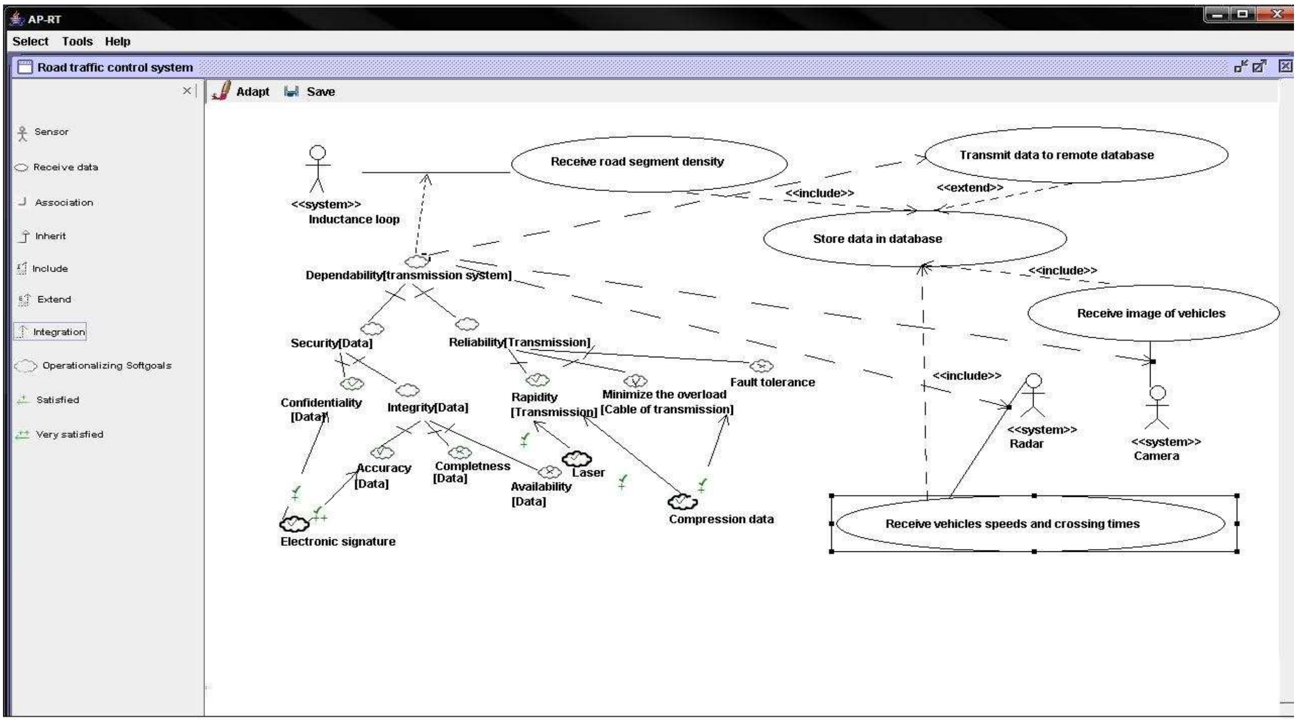


Figure 4. Data acquisition analysis model of Road traffic control system.

These functional requirements constitute the instances of the general use case “Receive data from the environment”. Furthermore, “VPN protocol” and “Noise reducing processing” constitute the operationalizing softgoals of the non functional requirements fulfilled by data transmission of MEDIVILLE system, which are: confidentiality and accuracy.

Reuse of the “Control of the environment” pattern: Figure 6 represents the control of patient information analysis model reusing the pattern “Control of the environment”. In fact, the “Controller” and “Control the environment” pattern’s elements are instantiated respectively by “Medical surveillance team” actor of MEDIVILLE system and “Control the patients at home” use case. Moreover, the in-door base station of MEDIVILLE system can detect patient’s falls and heart problems. Thereby, “Control the patients at home” use case can be extended by “detect falls” and “detect heart problems” use cases. These latter correspond to the instantiation of the pattern’s use case “detect the non-compliance of boundary constraints”.

Figure 6 represents the non-functional requirements that are fulfilled by the control system of the medical telesurveillance application. Confidentiality, rapidity of response time and reliability represent components of the quality of services offered by the MEDIVILLE system. These properties are ensured respectively by (i) “password” Operationalizing Softgoal, that allows to ensure the authentication of the medical surveillance members, (ii) “WS-DSAC mechanism”, that allows to perform admission control and load-balancing on a distributed platform, (iii) and “replication of processors”, that allows to prevent fault and then to ensure reliability of the system.

V. CONCLUSION

The main objectives of our work, described in this paper, are the definition of three analysis patterns and their reuse in RT applications. The first pattern represents the functional aspects of data acquisition system as well as the non-functional aspects of RT data transmission system. The second pattern shows the different kinds of anomalies that can be identified when time constraints related to the validity of data and deadline of transactions are not fulfilled. These anomalies are represented respectively by the use case “Detect the non freshness of data” and the use case “Detect the non-compliance of deadline of actions. The third pattern describes the different recovery actions that are activated by actuators. These patterns aim to facilitate the specification of RT applications analysis models and to improve their quality since they capture both past experience and best practices of RT systems designers.

Our future works include: (1) the definition of analysis patterns composition techniques in order to create a generic model that describes the common and the difference functions between RT applications, (2) the integration of the analysis patterns in the context of model driven architecture in order to add more assistance when defining the relationship between the proposed RT analysis and design patterns.

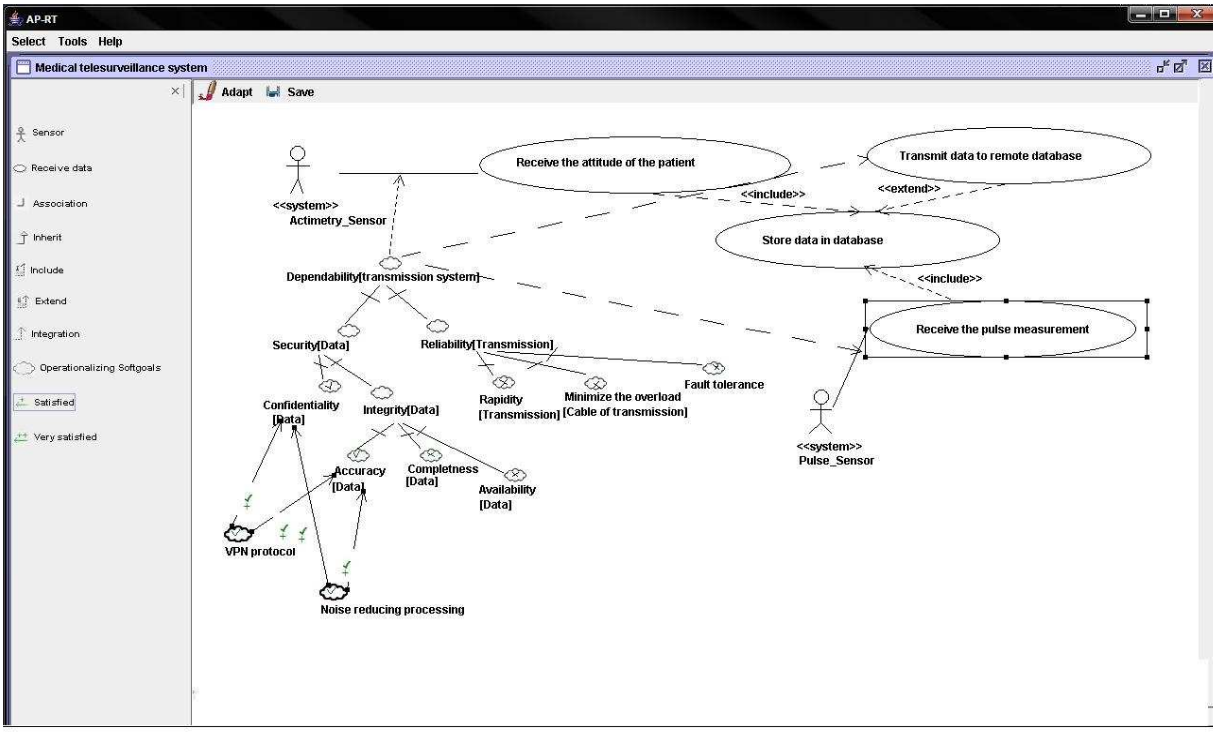


Figure 5. Data acquisition analysis model of medical telesurveillance system

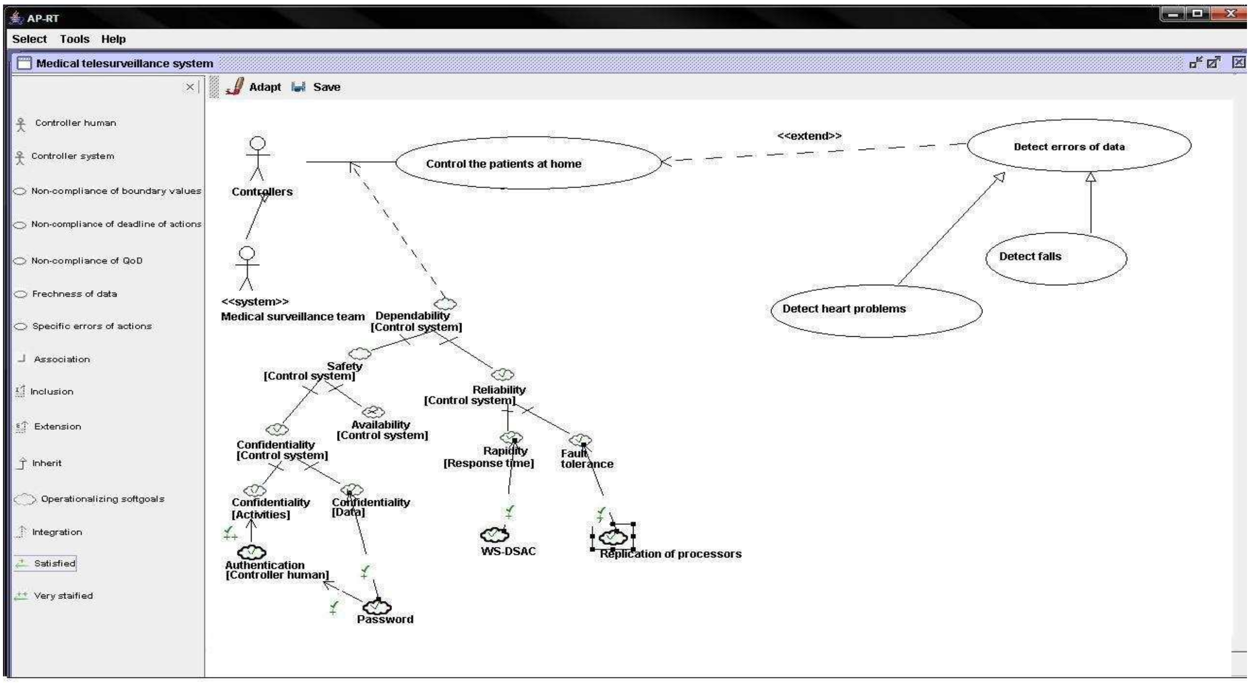

AP-RT
Select Tools Help
Medical telesurveillance system
Adapt Save
Controller human
Controller system
Non-compliance of boundary values
Non-compliance of deadline of actions
Non-compliance of QoD
Freshness of data
Specific errors of actions
Association
Inclusion
Extension
Inherit
Operationalizing softgoals
Integration
Satisfied
Very staified
Controllers
<<system>>
Medical surveillance team
Control the patients at home
<<extend>>
Detect errors of data
Detect falls
Detect heart problems
Dependability [Control system]
Safety [Control system]
Reliability [Control system]
Availability [Control system]
Confidentiality [Control system]
Rapidity [Response time]
Fault tolerance
Confidentiality [Activities]
Confidentiality [Data]
Authentication [Controller human]
Password
WS-DSAC
Replication of processors

Figure 6. Data control analysis model of medical telesurveillance system

This could bring new benefits and impulse for both the knowledge capturing techniques and the software development process quality.

REFERENCES

[1] Chung L. and Supakkul S., Representing NFRs and FRs: A goal-oriented and use case driven approach. *Software Engineering Research and Applications (SERA2004)*, LNCS, 3647(29-41), 2005.

[2] Konard S.J., Cheng B H.C. and Campbell L. A., "Object Analysis Patterns for Embedded Systems", IEEE Transactions on Software Engineering, Vol. 30, No. 12, December 2004.

[3] Jawawi D., Deris S. and Mamat R., Software Reuse for Mobile Robot Applications Through Analysis Patterns, The International Arab Journal of Information Technology, Vol. 4, No. 3, 2007.

[4] Fowler M., Analysis Patterns – Reusable Object Models, Addison-Wesley, 1997.

[5] Baldinger J.L., Boudy J., Dorizzi B., Levrey J.P., Andreao R., Perpère C., Delavault F., Rocaries F., Dietrich C. and Lacombe A., Tele-surveillance System for Patient at Home: The MEDIVILLE System, Book chapter in Computers Helping People with Special Needs, Springer Berlin, LNCS 3118, 2004.

[6] Fasel W., On-line traffic surveillance. Strasse and Verkehr revue, ISSN 0039-2189, vol 89, N°4, pp. 31-35, 2003.

[7] Stone R. and Wood K., Development of a Functional Basis for Design. Journal of Mechanical Design, 122(4): 359-370, 2000.

[8] Esfahani N., Mirian-Hosseinabadi S.H. and Rafati K., A Real-Time Analysis Process Patterns, Book chapter in Advances in Computer Science and Engineering, Springer-Verlag, CCIS 6, pp: 177-181, 2008.

[9] Chung L. and Sampaio J.C., On Non-Functional Requirements in Software Engineering, Book chapter in Conceptual Modeling: Foundations and Applications, Springer-Verlag, LNCS 5600, pp. 363–379, 2009.

[10] Chung L. and Supakkul S., Capturing and Reusing Functional and Non-functional Requirements Knowledge: A Goal-Object Pattern Approach, Proceedings of IEEE International Conference on Information Reuse and Integration, 10.1109/IRI.2006.252471, pp. 539 – 544, 2006.

[11] Tonu S.A., Incorporating Non-Functional Requirements with UML Models, Phd thesis presented to the University of Waterloo, Ontario-Canada, 2006.

[12] Amirijoo M., Hansson J., and Son S. H., Specification and management of QoS in real-time databases supporting imprecise computations. IEEE Transactions on Computers, 55(3), 2006.

[13] J. Laprie, "Dependable Computing and Fault-Tolerant Systems", Depndability: Basic Concepts and Terminology in Eng-lish, French, German, Italian and Japanese. Vol 5. Springer-Verlag, 1992.

[14] Serra A., Gaïti D., Barroso G. and Boudy J., Assuring QoS Differentiation and Load Balancing on Web Servers Clusters, Proceedings of the IEEE Conference on Control Applications, pp. 885 – 890, Toronto, Canada, August 28-31, 2005.

[15] OMG, UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms V 1.1, April 2008.

[16] Zhu L. and Gorton I., UML Profiles for Design Decisions and Non-Functional Requirements, International Conference on Software Engineering, Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, page 8, ISBN 0-7695-2951-8, 2007.