RESEARCH ARTICLE OPEN ACCESS

Selection and Reuse of Software Design Patterns Using CBR and Word Net

Debabrata Sahu, Samiksha Mahanta
Gandhi Institute of Excellent Technocrats, Bhubaneswar, India
Subas Institute of Technology, Bhubaneswar, Odisha, India

ABSTRACT
Softwareengineersandprogrammersdealwithrepeatedproblems and situations in the course of software design.This lead to the development of software design patterns,which can be defined as a description of an abstract so-lution for abstract design problems.Existing approachesto pattern application using computer tools, need the helpand guidance of a human designer to select which designpatterntoapply.Theautomationofthistaskopensthepos-sibility of CASE design tools providing complete automa-tion for the application of design patterns, and the offeringnew functionalities that can help the software designer toimprove systems, and do better software reuse. In this pa-per we present an approach that automates design patternselectionandapplication.ThisapproachisbasedonCase-BasedReasoningandWordNet,showinghowtheyarecom-bined to generate evolved software design diagrams.Wealsopresentanexperimentalstudyofourapproach.

I. INTRODUCTION

The complexity of software systems is increasing to theextent that software development teams have difficulty todeliver systems within the schedule accorded with clients.But this is not the only problem, complexity brings bugsand unforseen situations by the system specifications. Oneway to attenuate this problem is to reuse software [11, 5],not only code, but other knowledge gathered during thesoftware development process.Among the different typesof knowledge involved in the software development, is theknowledge about prototypical situations, and how they canbe efficiently solved. In the software engineering researcharea, the development of software design patterns [7] hadthe goal of cataloging these common situations that appearinmostofallsoftwaresystems.

Softwaredesignpatternsareanelegantandefficientwayof solving abstract problems. Each pattern describe a solu-tion thatcomprisesaset ofstepsthatcan beusedtodesign a specific part of the system being developed.They con-dense knowledge about how the problem should be viewedand what are the consequences of using a specific pattern.The development of these patterns were mainly for humanusage, but there were efforts from the research communityto automate the application of design patterns. Most of thedevelopedapproachesneedhumanintervention,atleastforselection of the pattern to be applied. In this paper we pro-poseanapproachthatautomatescompletelytheapplicationofsoftwaredesignpatterns.

There are several research works that have common as-pects with our approach.Eden et. al. [6] has proposedan approach to the specification of design patterns, and aprototype tool that automates extensively their application.This approach considers design patterns as programs thatmanipulate other programs, thus they are viewed as meta-programs. Eden’s approach does not automates all the pro-cessofdesignapplication,sinceitistheuserthathastose-lect which pattern to apply. Another important issue is theabstractionlevelofapplication,whichinEden’scaseisthecode level, while in our approach is the design level. Tokudaand Batory [14] also present an approach in which patternsare expressed in the form of a series of parameterized pro-gramtransformationsappliedtosoftwarecode.LikeEden’swork, this work does not address the automation owhichpatterntoapply.Otherworksonspecifyingdesignpatternsandautomatingitsapplicationarepresented byBär[3]and Cinnéide[4].Theseworksalsoautomatetheapplication of design patterns, but do not select which pattern to ap-ply, this is done by the user. Both works deal with designmodificationinsteadofcodemodification.Guéhe´neucand Jussien [8] developed an application of explanation-basedconstraintprogrammingfortheidentificationofdesignpat-ternsinobject-orientedsourcecode. Ourapproachisbasedontheideathatasystemcanlearnto select and to apply design patterns if it can store andreuse experiences that encode the situation in which pat-terns are used.This idea is based on an area of ArtificialIntelligence [13] called Case-Based Reasoning (CBR, [9]).CBRisbasedontheideanotonlyofreasoningfromexperi-

ence, but also to learn from it. If the application of a specificsoftware design pattern can be represented in the form of acase, then CBR can be used for the automation of designpatterns.Our approach considers a case to be a situationwhere a design pattern was applied to a specific softwaredesign(intheformofUnifiedModellingLanguage-UML [12] class diagram). Cases are stored in a case library andindexed using a general ontology (WordNet [10]). The CBRframework that we propose selects which pattern to apply,regarding the target design problem, generating a new de-sign.It can also learn new cases from the application ofdesign patterns.This approach has been implemented inREBUILDER, a CASE tool which provides new function-alitiesbasedonCBR.

There are two fundamental concepts that need to be fur-ther explored: software design patterns detailed in section2, and CBR described in section 3.Then we present theREBUILDERtool,whichintegratesourapproach,followedby the detailed description of our approach. Finally we de-scribe the experimental work made with REBUILDER us-ingourapproachinsection7andconcludeinsection8.

Software Design Patterns

Asoftwaredesignpatterndescribesasolution foranab-stract design problem.This solution is described in termsof communicating objects that are customized to solve thedesign problem in a specific context. A pattern descriptioncomprisesfourmainelements:

Nameisthedescriptionwhichidentifiesthedesignpattern,andisessentialforcommunicationbetweendesigners.

Problem describes the application conditions of the designpattern and the problem situation that the pattern in-tendstosolve.Italsodescribestheapplicationcontextthroughexamplesorobjectstructures.

Solution describes the design elements that comprise thedesignsolution,alongwiththerelationships,responsi-bilities and collaborations. This is done at an abstractlevel,sinceadesignpatterncanbeappliedtodifferentsituations.

Outcome describes the consequences of the pattern appli-cation.Mostofthetimespatternspresenttrade-offstothedesigner,whichneedtobeanalyzed.

Eric et al.[7] describe a catalog comprising 23 designpatterns, and give a more detailed description for each pat-tern consisting on: pattern name and classification, patternintent, other well-known names for the pattern, motivation,applicability,structure,participants,collaborations,con-sequences,implementationexample,samplecode,known uses,andrelatedpatterns.Fromtheseitems,wedrawatten-tion to the participants and the structure.The participantsdescribe the objects that participate in the pattern, alongwith their responsibilities and roles. These objects play animportant role in our approach.The structure is a graphi-cal representation of the design pattern, where objects andrelationsbetweenthemarerepresented.

A Pattern is classified based on its function or goal,whichcategorizespatternsas:creational,structural,andbe-havioral.Creational patterns have the main goal of objectcreation, structural patterns deal with structural changes,and behavioral patterns deal with the way objects relate witheachother,andthewaytheydistributeresponsibility.

As an example of a design pattern we briefly present theAbstract Factory design pattern (see [7], page 87). The in-tent of this pattern is to provide an interface for creation offamilies of objects without specifying their concrete classes.Basically there are two dimensions in objects: object types,and object families.Concerning the type of objects, eachtype represents a group of objects having the same concep-tual classification, like window or scrollbar. The family ofobjects defines a group of objects that belong to a specificconceptual family, not the same class of objects.For ex-ample,MotifobjectsandMSWindowobjects,whereMotifobjects can be windows,scrollbars or buttons,which ex-ist in MS Window objects but do not have the same visualcharacteristics.

Suppose now, that an user interface toolkit is being im-plemented. This toolkit provides several types of interfaceobjects, like windows, scroll bars, buttons, and text boxes.The toolkit can support also different look-and-feel stan-dards, for example, Motif, MS Windows, and Macintosh.In order for the toolkit to be portable, object creation mustbeflexibleandcannotbehardcoded.Asolutiontotheflex-iblecreationofobjectsdependingonthelook-and-feel,canbe obtained through the application of the Abstract Factorydesign pattern.This pattern has five types of participatingobjects:

- The Abstract Factory object declares an interface foroperationsthatcreateabstractproducts.
- Concrete Factory objects implement the operations tocreateconcreteproducts.

AbstractProductobjectsdeclareaninterfaceforatypeofproductobject.
ConcreteProductobjectsdefineaproductobjecttobecreatedbythecorrespondingconcretefactory,andalsoimplementtheAbstractProductinterface.
ClientobjectsuseonlyinterfacesdeclaredbyAbstractFactoryandAbstractProductclasses.

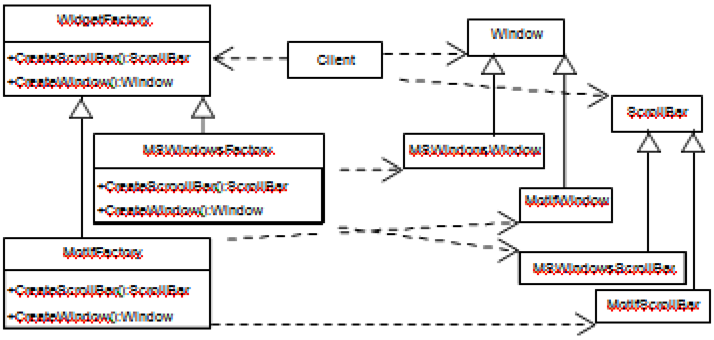

Figure1.TheapplicationoftheAbstractFac-tory design pattern to the interface toolkitproblem.

A possible solution structure for the problem posed bytheinterfacetoolkitisdepictedinfigure1.Thepat-ternparticipantsare:abstractfactory(WidgetFactory), con-cretefactories(MSWindowsFactoryandMotifFactory),ab-stract products (Window and ScrollBar),concrete prod-ucts(MSWindowsWindow,MotifWindow,MSWindowsS-crollBar, and MotifScrollBar), and client (Client). The cre-atemethodsinthefactoriesaretheonlywaythatclientscancreatetheinterfaceobjects,thuscontrollingandabstractingobject creation.The main consequences of this pattern isthat it isolates concrete classes, makes exchanging productfamilieseasy,andpromotesconsistencyamongproducts.

Case-BasedReasoning

Case-BasedReasoningcanbeviewedasamethodologyfordevelopingknowledge-basedsystems[2]thatmakesuseof experience for reasoning about problems. Its main ideaistoreusepastexperiencestosolvenewsituationsorprob-lems.

A case is a central concept in CBR, and it represents achunk of experience in a format that can be reused by a CBRsystem.Usually a case comprises three main parts: prob-lem, solution, and outcome.The problem is a descriptionof the situation that the case is representing.This can be,for example, the symptoms of a patient in case of a med-ical situation, or a software system's requirements, or anydescription that can characterize the situation being repre-sented. The solution describes what was used to solve thesituation described in the problem. For instance, in the med-icaldomainitcanbethetreatmentsusedtohealthepatient,or in the software domain a design that complies with thesystem'srequirements.Theoutcomeexpressestheresultofthe application of the solution to the problem. This meansthatcommonlytherearetwopossibleoutcomes: successorfailure.A success case represents a situation in which thesolutionworkedwell,whileafailurecaserepresents asit-

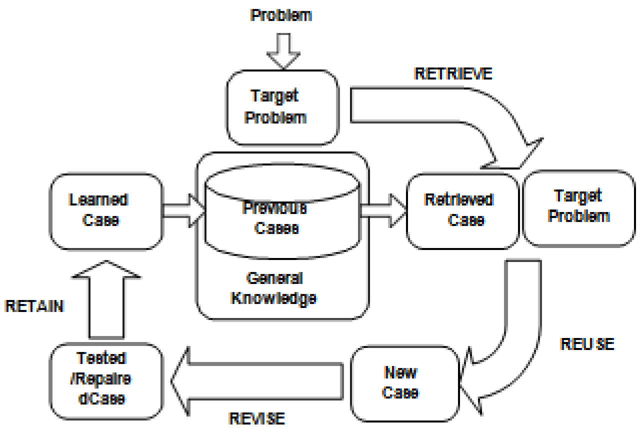

Figure2.TheCBRcycle.

uation where the solution did not work. There can be otherpartsofcaseslikethejustificationthatrelatesproblemwithsolutionthroughcausalrelations.

Another important part of the CBR methodology is thecase library. This is the place where all the cases are goingtobestoredandorganized.Duetothehighnumber ofcasesthat the library can have, most of the CBR systems use in-dexing structures that enable fast retrieval of relevant casesfrom memory. So most of the times a case library is morethanjusttheplacetostorecases,butitdefineshowtheyarestoredandhowtheycanbeaccessed.

At an abstract level CBR can be described by the rea-soning cycle depicted in figure 2 [1].The reasoning pro-cess starts with the problem description, which is then trans-formed into a target problem (or query case). The problemisprovidedbyasystemuserorbyanothersystem.Thefirstphase in the CBR cycle is to retrieve from the case librarythe cases that are relevant for the target problem. The rele-vancyofacasemustbedefinedbythesystem,butthemostcommondefinitionisbasedonfeaturesimilarity.Inthe endof retrieval, the best retrieved case[1] is returned and passestothenextphasealongwiththetargetcase.

The reuse phase (also designated as adaptation phase)adapts the retrieved case to the target problem, yielding asolved case (or new case). This process can be performedwithseveralinferencetechniques,andmany workhasbeendone on the subject [15]. The next step for a CBR sys-temistorevisethenewcasereturningatestedandrepairedcase. This phase usually comprises two parts: verificationandevaluation.Whileverificationcheckstthenewcasecon-sistency and coherence, the evaluation phase assesses theperformancecharacteristicsofthenewcase.Finally, there-tain phase learns the solved case by storing it in the caselibrary. This phase is more complex than it seems, becausenotallcasesshouldbestored.Ifanewcaseisequalorvery

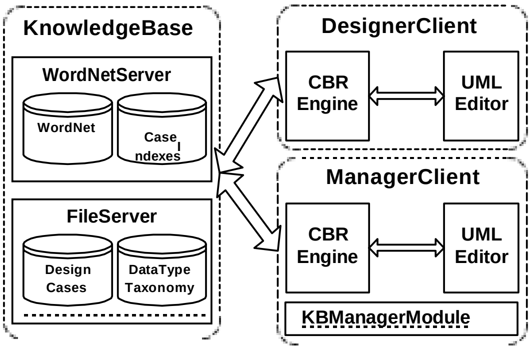

Figure3.REBUILDER'sArchitecture.

similartoacasealreadyinthelibrary,thenitshouldnotbe stored because it brings nothing new to the system and itdegrades the system's performance. This last phase closestheCBRcyclebyfeedingthesystemwithnewexperiences,makingthesystemcapableoflearning.

REBUILDER

REBUILDER has two main goals:centralize the cor-poration's design knowledge, and provide the software de-signer with a design environment capable of promotingsoftware design reuse.This is achieved in our approachwithCBRasthemainreasoningprocess,and withcasesas the main knowledge pieces. This section describes RE-BUILDER, detailing it's architecture, knowledge base andreasoningengine. REBUILDER is based on a client-server architecturecomprising two servers and two clients (see Figure 3). Theknowledge base (KB) used in REBUILDER comprises theWordNet server and the file server, while the clients com-prise similar modules. The main difference between clientsis that the manager client has an extra module allowing theKB maintenance. There can only be one server of each type,and only one manager client. Designer clients can be sev-eraldependingonhardwareresources.

The WordNet server comprises the WordNet ontologyand the Case Indexes. WordNet is a general ontology usedin REBUILDER to index cases using semantics. It also en-ables the assessment of semantic similarity between con-cepts, used in REBUILDER for case similarity.The caseindexes are used for fast retrieval of cases from the case li-brary. These indexes are associated to cases and to piecesofcasesalso,enablingflexibleretrieval.

Clients request the file server for cases, which are in acentralized repository called case library.Each file repre-sents an UML design (see figure 4). This enables the clientto work only with the strictly necessary cases. For the opti-mizationofthisprocessthecaseindexesplayacrucialrole.
Thedatatypetaxonomyisusedforcomparingdatatypes,and is a very simple taxonomy of the main Java data types.TheUMLeditor,theKBmanagermoduleandCBRen-gineconstitutethemanagerclient(thedesignerclientisequaltothisclientexceptthatitdoesnothavethemanager module).
The UML editor is the front-end of REBUILDER andthe environment where the software designer develops de-signs.Apartfromtheusualeditorcommandstomanipulate UML objects, the editor integrates new commands capableofreusingdesignknowledge.Thesecommandsaredirectlyrelatedwiththe CBRenginecapabilities.
TheKBmanagermoduleisusedbytheadministratorto manage the KB, keeping it consistent and updated.Thismodulecomprisesallthecase-basemanagementfunctions.TheseareusedbytheKBadministratortoupdateandmod-ifytheKB.
TheCBREngineisthereasoningmoduleofRE-BUILDER. This module comprises six different parts: Re-trieval,Design Composition,Analogy,Design Patterns,Verification, and Learning.The Retrieval sub-module re-trieves cases from the case library based on the similaritywith the target problem.The Design Composition sub-module modifies old cases to create new solutions.It cantakepiecesofoneormorecasestobuildanewsolutionby composition of these pieces.The Design Patterns sub-module, uses software design patterns and CBR for gener-ation of new designs.Analogy establishes a mapping be-tween problem and selected cases, which is then used tobuild a new design by knowledge transfer between the se-lectedcaseandthetargetproblem.CaseVerificationchecksthe coherence and consistency of the cases created or mod-ified by the system. It revises a

solution generated by RE-BUILDER before it is shown to the software designer. Thelast reasoning sub-module is the retain phase, where the sys-tem learns new cases. The cases generated by REBUILDERare stored in the case library and indexed using a memorystructure.

CBRApproachtoSoftwareDesignPatterns

This section presents how software design patterns canbe applied to a target design using CBR. We start by de-scribing the patterns module, and then we describe each ofitspartsinmoredetail.

Architecture

Figure 5 presents the architecture of the patterns mod-ule.It comprises three phases: retrieve applicable DesignPatternApplication(DPA)cases,selectbestDPAcase,andapply selected DPA case. A DPA case describes the appli-cation ofaspecificdesignpatterntoasoftwaredesign(the

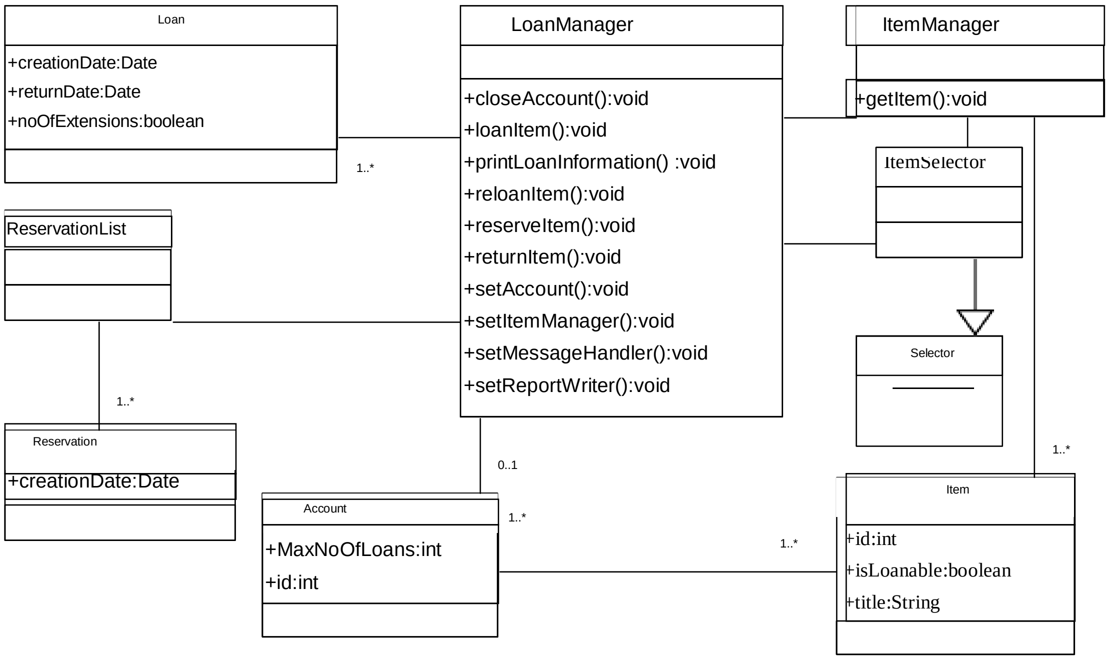

Figure4.ExampleofanUMLclassdiagram.

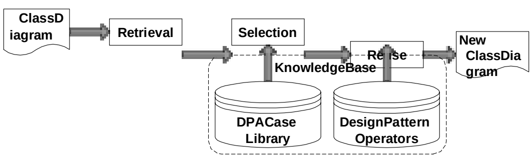

Figure5.Softwaredesignpatternapplicationmodule.

nextsubsectiondescribesthecaserepresentationindetail).This module is used when the user decides to apply designpatternstoimprovethecurrentdesign.

The first phase uses a target class diagram as the prob-lem,andsearchestheDPAcaselibraryforDPAcasesthat match the problem. Then the retrieved DPA cases areranked and the best one is selected for application,whichis performed in the next step. The application of the DPAcaseusesthedesignpatternoperatorsandyieldsanewclassdiagram, which is then used to build a new DPA case. ThisnewcaseisstoredintheDPAcaselibrary.

DPACaseRepresentation

A DPA case describes a specific situation where a soft-ware design pattern was applied to a class diagram.EachDPA case comprises: a problem and a solution description.The problem describes the situation of application based on:the initial class diagram, and the mapped participants. TheinitialclassdiagramistheUMLclassdiagramtowhichthesoftware design pattern was applied.Like the name indi-cates, it is the pre-modification diagram. The mapped par-ticipants are specific elements that

must be present in orderforthesoftwaredesignpatterntobeapplicable.Participantscanbe:objects, methodsor attributes.Each participanthas a specific role in the design pattern and it is important forthe correct application of the design pattern.Each patternhasit'sspecificsetofparticipants.Oncetheparticipantsareidentified the application of a design pattern follows a spe-cific algorithm that embeds the pattern actions.Mappingtheparticipantsisperformedtoselect aroleforsomeoftheobjects,attributesand/ormethodsininitialclassdiagram.
It is important to describe the types of participants de-fined in our approach. Object participants can be classesor interfaces, attribute participants correspond to class at-tributes, and method participants correspond to object meth-ods.Eachparticipanthasasetofproperties:

Role(String) Roleoftheparticipantinthedesignpattern.
Object (class or interface) Object playing the role, or incase of attribute or method participant the object towhichtheattributeormethodbelongs.
Method (method) Method playing the role in case of amethodparticipant.
Attribute (attribute) Attribute playing the role in case ofanattributeparticipant.
Mandatory (Boolean) Optional or not, if the participantmustexistinorderforthedesignpatterntobe applica-ble,orjustoptional.
Unique (Boolean) Unique or not, if there can be one ormoreparticipantsoftheRoletype.
The solution description of a DPA case is the name of thedesign pattern applied, which is then used to select the cor-respondentsoftwaredesignpatternoperator.Different DPAcasescanhavethesamesolution,becausewhataDPAcaserepresents is the context of application of a design pattern,andthereareinfinitecontextsituations.

DPACaseLibrary

The DPA cases are indexed using the context synsets ofthe object participants (see figure 6) and only the partici-pants (objects, attributes and methods) can be used as re-trieval indexes. The WordNet structure is used as an indexstructure enabling the search for DPA cases in an incremen-tal way.Each case can be stored in a file, which can beread only when needed. In figure 6 there are four indexedobjects, three of them corresponding to object participants,and one a method participant, indexed by the object com-prisingthemethod.

Software Design Pattern Operators

For each design pattern there is one operator, for in-stance,theAbstractFactorydesignpatternhasaspecific

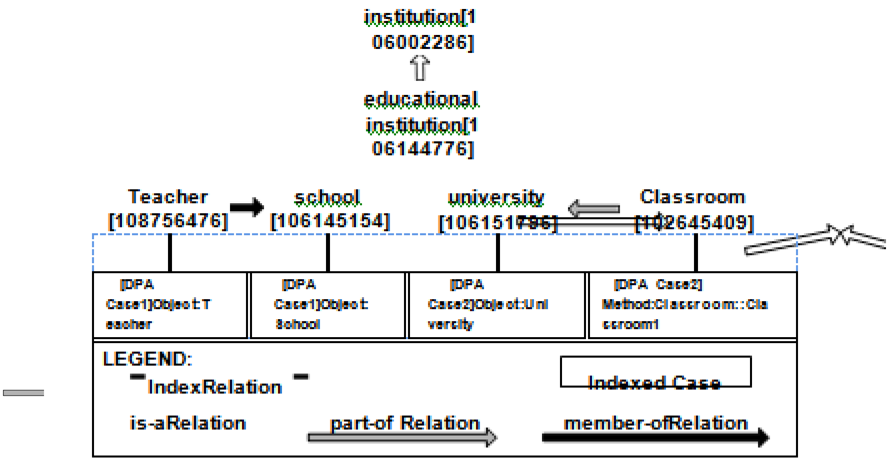

Figure 6.An example of the DPA case index-ing.

pattern operator, which defines how to apply the AbstractFactory pattern, and if it can be applied. A software designpattern operator comprises three parts:the set of specificparticipants, the application conditions, and the actions forathespecificdesignpattern.
The participants are key objects, methods or attributesthat play an important and active role in a design pattern.For example, the participants specification for the AbstractFactorypatternoperatorare:
AbstractFactory(**Type**: Object; **Mandatory**: no; **Unique**: yes): De-claresaninterfaceforoperationsthatcreateabstractproductobjects.
ConcreteFactory(**Type**:Object;**Mandatory**:no;**Unique**:no):Im-plementstheoperationstocreateconcreteproductobjec

ts.

- AbstractProduct(**Type**: Object; **Mandatory**: no; **Unique**: no): De-claresaninterfaceforatypeofproductobject.
- ConcreteProduct(**Type**:Object; **Mandatory**:yes; **Unique**:no):Definesaproductobjecttobecreatedbythecorrespondingconcretefactory.
- Client (**Type**:Object; **Mandatory**:no; **Unique**:no):Uses onlyinterfacesdeclaredbyAbstractFactoryandAbstractProductclasses.

Application conditions define the constraints that mustbe met by participant objects in order to the operator to beapplied.In the case of the abstract factory the applicationconditions are: there must be at least one ConcreteProduct,andthatallConcreteProductsmusthavenopublicattributesorstaticmethods.

The pattern actions for Abstract Factory are defined inthe algorithm 7.This algorithm transforms ClassDiagraminto NewClassDiagramthrough the application of the Ab-stractFactorydesignpattern.

Retrieval of DPA Cases

The retrieval of DPA cases is done using the WordNetas the indexing structure. The retrieval algorithm (see fig-ure8)startswiththetargetclassdiagram(ClassDiagram).

```
NewClassDiagram←CopyClassDiagram
ConcreteProducts←Getallconcreteproductsfromthe
                 mappingdonebetweentheselectedcase
                 andtheproblem(Role=ConcreteProduct)
Clients←GetallclassesthatuseatleastoneConcreteProduct
AbstractFactory←Createnewclasswith
                 name"AbstractFactory"
AddtoNewClassDiagramtheclassAbstractFactory
AbstractProducts←ForeachConcreteProductget/createthe
         correspondentAbstractProduct
ConcreteFactories←Get/createalltheConcreteFactoriesfor
                   ConcreteProducts
FOREACHProductinConcreteProductsDO
         AbstractProduct←GetProductsuperclass
         AbstracttheaccessofClientsfromProductto
                 AbstractProduct
         EncapsulatetheconstructionofProductin
                 AbstractFactoryandAbstractProduct
ENDFOR
ApplytheSingletonpatterntoAbstractFactoryRE
TURNNewClassDiagram
```

Figure7.TheapplicationalgorithmfortheAb-stractFactorydesignpattern.

REBUILDER only uses UML class diagrams for reason-ing tasks, the other UML diagrams are not used as queries.Then ituses thecontext synsets ofthe objects inthe tar-getdiagramasprobestosearchtheWordNetstructure.Thealgorithminitiatesthesearchonthesynsetprobes,and then expands the search to neighbor synsets using allthe WordNet semantic relations (is-a, part-of, member-of,andsubstance-of).Itrunsuntilthenumberofcasestoberetrieved(NumberOfCases)isreached,orthemaximumsearch level (MSL) is reached, or the Synsetslist is emptywhichcorrespondstotheexhaustivesearchoftheWordNet.TheDPAcaseretrievalalgorithmreceivestheinputparam-eters: ClassDiagram, NumberOfCases, and MSL; returningasetofretrievedcases(SelectedCases).

SelectionofDPACases

After the retrieval of the relevant cases, they are rankedaccordinglytoitsapplicabilitytothetargetdiagram(Class-Diagram). The selection algorithm (see figure 9) starts bymapping the ClassDiagramto each of the retrieved cases(SelectedCases), resulting in a mapping for each case. Themapping is performed from the case's participants to thetarget class diagram (only the mandatory participants aremapped).Associated to each mapping there is a score,which is given by the number of mapped participants. So,what this score measures is the degree of participants map-pingbetweentheDPAcaseandthetargetdiagram.

The next step in the algorithm is to rank the

Selected- Caseslistbasedonthemappingscores.Thefinalphase

```
Scores← ?
Mappings←?
FORALLSelectedCaseinSelectedCasesDO
        Mapping/Score←Getthemappingandscorefor
                    theSelectedCase
        Add to Mappingsthe
        SelectedCaseMappingAddto Scoresthe
        SelectedCaseScore
ENDFOR
Rank lists:SelectedCases, Mappingsand Scores, by
ScoresFORALLSelectedCaseinSelectedCasesDO
        IF(DesignPattern(solutionofSelectedCase)canbeapplied
          toClassDiagramusingtheMappingestablishedbefore)T
          HEN
          RETURNSelectedCaseandtherespectiveMappingEN
        DIF
ENDFORRETUR
```

Figure 9.The algorithm for selection of DPAcases.TheinputlistofDPAcasesisSelected-Cases.

```
Objects←GetallclassesandinterfacesfromtheClassDiagram
Synsets←?
FORALLObjectinObjectsDO
        AddtoSynsetslisttheObject'scontextsynset
ENDFOR
SelectedCases←?
SearchLevel← ?
Explored←?
WHILE  (#SelectedCases<NumberOfCases)AND
                    (SearchLevel<MSL)AND
                    (Synsets≠)DO
        DPACases←GetallDPAcasesindexedbyatleastone
                    synsetfromSynsets
        AddtoSelectedCasestheDPACaseslist
        NewSynsets←?
        FORALLSynsetin SynsetsDO
                Neighbors←GetSynsethypernyms,
                        hyponyms,holonymsandmeronyms
                Neighbors←Neighbors−Synsets
                        −Explored−NewSynsets
                AddtoNewSynsetstheNeighborslist
        ENDFOR
        AddtoExploredtheSynsetslist
        Synsets←NewSynsets
        SearchLevel←SearchLevel+1
ENDWHILE
RETURNSelectedCases
```

Figure8.TheretrievalalgorithmforDPAcases.

consistsoncheckingtheapplicabilityofthebestDPAcase,which is done using the design pattern operator associatedwith the DPA case. If the application conditions of this op-eratorarenotviolated,thenthisDPAcaseisreturnedastheselected one. Otherwise, this case is discarded and the nextbest case goes through the same process, until one applica-blecaseisfoundoritreturnsnull.

ApplicationofDPACases

Selected the DPA case, the next step is to apply it to thetarget class diagram generating a new class diagram and anew DPA case. Other UML diagrams are not used for rea-soning purposes, thus are not changed.The application ofa DPA case is done using the pattern operator correspond-ing to the software design pattern given as the solution ofthe DPA case.Starting with the participants mapping es-tablishedbefore,theapplicationofthepatternisdoneusingtheapplicationalgorithmofthepatternoperator.

ExperimentalWork

Experimentswereperformedtoevaluatetheperfor-mance of our approach.We used a case-base of 60 DPAcases,eachonerepresentinganapplicationofasoftware
design pattern to an UML class diagram.Each DPA casewas generated from a different class diagram.For theseexperiments five software design patterns were used, thenames of the patterns accordingly to [7] are: Abstract Fac-tory, Builder, Composite, Singleton and Prototype.EachofthesepatternsareimplementedREBUILDERalongwiththeparticipantsdefinitionandoperators.

We also defined 25 test class diagrams to evaluate theprecisionoftheretrievalmechanism.Thesediagramscom-prisethreetofiveobjectsandhavenomethodorattributes.For each test diagram the algorithm retrieved 15 DPA cases,which were then evaluated.This evaluation consisted indefiningifthepatternsandparticipantschosenwerecorrector if they were not applicable. The results are presented infigure10.

The precision results show that the retrieval mechanismfor this set of problems achieves 76% of correct selectedDPA cases with a retrieval set size of three (cumulative re-sult), which is in our opinion a very good indicator.Asexpected, the non-cumulative results degrade with the in-creaseintherankofretrievedcases.Thisalsoindicatesthatthesimilaritymetricusedtoranktheretrievedcasesis per-forming as desired, choosing the best cases for the initialplaces of the ranking. The cumulative values show that theprecisionrangesfrom76%(retrievalsetsizeof3and 4),to39%(retrievalsetsizeof15).

II. CONCLUSIONS

This paper presents an approach to the selection andapplication of software design patterns in an automatedway.UsingCBRandWordNetweareabletostoresit-

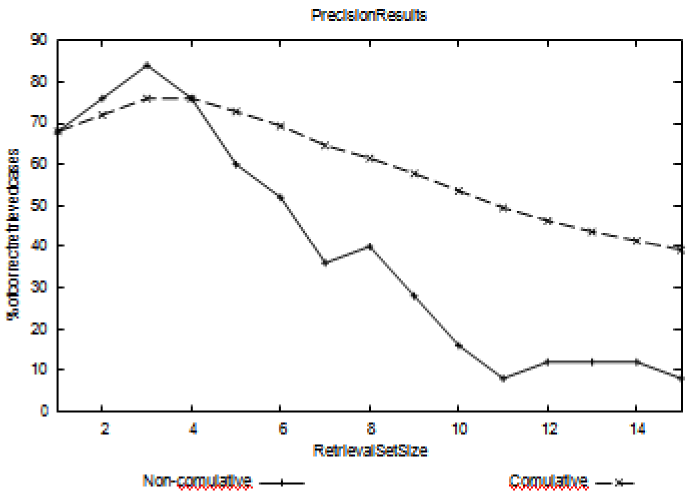

Figure10.TheprecisionvaluesfortheDPAretrievalalgorithm.

uations where design patterns were applied.These situa-tions, called cases, can then be reused in similar situationsto guide design pattern selection and application. This ap-proach was implemented and tested in a CASE tool namedREBUILDER,whichusesUMLtomodelsoftware systems.

An obvious advantage of our approach is the completeautomation of the application of design patterns.Our ap-proach selects which pattern to apply based on DPA cases.This enables a CASE tool to offer new functionalities,aimed for design maintenance and reuse.For instance, itcansuggesttothesoftwaredesignerseveraldesignalterna-tivesbasedontheapplicationofdifferentdesignpatterns.

One limitation of our approach is that the system per-formance depends on the quality and diversity of the caselibrary, which will improve as time follows. Another lim-itation, is that the range of case application is always re-stricted, and it does not outperform a software designer abil-itytoidentifywhichpatterntoapply.Despitethis,wethinkthat our approach can provide a good contribute for designimprovement, especially in situations when the user has todeal with a huge amount of objects.In this situation, au-tomation is possibly the only way to apply design patterns,since it is difficult for the designer to deal with such anamountofobjects.

REFERENCES

[1] A. Aamodt and E. Plaza.Case–based reasoning:Foun-dational issues, methodological variations, and system ap-proaches.AICommunications,7(1):39–59,1994.
[2] K.-D.Althoff. Case-basedreasoning. InS.K.Chang,editor,Handbook on Software Engineering and Knowledge Engi-neering,volume1,pages549–588.WorldScientific,2001.
[3] H.Bär,M.Bauer,O.Ciupke,S.Demeyer,S.Ducasse,M. Lanza, R. Marinescu, R. Nebbe, O. Nierstrasz, T. Rich-ner,M.Rieger,C.Riva,A.-M.Sassen,B.Schulz,P. Steyaert, S. Tichelaar, and J. Weisbrod.The famoosobject–orientedreengineeringhandbook.Technicalre-port,ForschungszentrumInformatik,SoftwareCompositionGroup,UniversityofBerne,1999.
[4] M.CinnéideandP.Nixon.Amethodologyforthe auto- mated introduction of design patterns.In IEEE Interna-tional Conference on Software Maintenance, Oxford, Eng-land,1999.IEEE.
[5] B. Coulange.Software Reuse.Springer Verlag, London,1997.
[6] A.Eden,J.Gil,andA.Yehudai.Automatingtheapplicationof design patterns.Journal of Object Oriented Program-ming,(May),1997.
[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides.DesignPatterns: Elements of Reusable Object-Oriented Software.Addison-Wesley,Reading,1995.
[8] Y.-G.GuéhéneucandN.Jussien.Usingexplanationsforde- sign patterns identification. In IJCAI'01 Workshop on Mod-ellingandSolvingProblemswithConstraints,pages57–64,Seattle,WA,USA,2001.
[9] J. Kolodner.Case-Based Reasoning.Morgan Kaufman,1993.
[10] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J.Miller.Introductiontowordnet:anon-linelexicaldatabase.InternationalJournalofLexicography,3(4):235–244,1990.
[11] R. PrietoDiaz.Status report:Software reusability.IEEESoftware,3(May),1993.
[12] J.Rumbaugh,I.Jacobson,andG.Booch.TheUnifiedMod-elingLanguageReferenceManual.Addison-Wesley,Read-ing,MA,1998.
[13] S. Russel and P. Norvig.Artificial Intelligence: A ModernApproach.PrenticeHall,NewJersey,1995.
[14] L.TokudaandD.Batory.Automatedsoftwareevolutionviadesignpatterns.In3rdInternationalSymposiumonAppliedCorporateComputing, Monterrey,Mexico,1995.
[15] A. Voss.Towards a methodology for case adaptation.InECAI '96; 12th European Conference on Artificial Intelli-gence, pages 147–154, Chichester - New York - Brisbane,Aug.1996.Wiley.