FAULT DETECTION OF CONSTRAINED OBJECT MODELS USING OBJECT MODELING TECHNIQUES

Priya R
Assistant Professor
Department of Computer Science, Government College
Kariavattom, Trivandrum 695 581, Kerala, India
priyanil2007@gmail.com

Abstract - This paper proposes a new efficient methodology for fault detection from the structural modeling of a complex system using the UML class diagram. The crisp and precise constraints on the attributes and relationships are represented using the formal specification language, Object Constraint Language (OCL). Model verification is done by converting the model to Constraint Satisfaction Problem(CSP) and by solving the problem. Performance is analysed by both bounded and unbounded validation.

***Keywords*:** Complex system; constraints; model verification.

1. Introduction

Object Modeling Technique (OMT), developed by Rumbaugh *et al.* in 1991 is an object modeling approach using objects and their behaviour, which helps to reduce the complexity in developing a complex system. The purpose of OMT is mainly testing physical entities before building them by simulating the object model, dynamic model and functional models. OMT also helps to visualize the system so that the complexity involved in building the system can be reduced by using Unified Modeling Language (UML) which is a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system.

Class diagram, which is a structural UML diagram focuses on the static aspects of the system. Class diagrams typically specify the main elements of the entities of the system, but fail to provide all the relevant aspects of the specification. Designers usually need to add constraints to the model to cater to this weakness of class diagrams. These constraints are often described in natural language. However, constraints written using natural language inevitably lead to ambiguities, and hence Object Constraint Language (OCL), which is a formal specification language developed by IBM, could be employed. OCL offers three types of constraints on class diagrams with a fourth constraint offering the user the ability to extend the class diagram. They are *Pre-condition, Post-condition, Invariant* and *Definition.*

Modeling of complex systems is relevant since a complex system consists of many components, with numerous relationships and interactions between these components and the components will be linked through many dense interconnections and they cannot be described by a single rule and their characteristics are not reducible to one level of description. This complexity leads to many errors that may creep in during modeling. This necessitates the Validation and Verification (V&V) of models. But, UML class diagrams annotated with OCL constraints is just a pictorial language used to make software blue prints and model verification and execution is not possible with the structural modeling of UML.

In this paper, a methodology is proposed to translate the Object oriented class diagram with its constraints to an intermediate representation where formal verifications could be performed. Formal validations are also employed on the class diagram before code generation, which would enable to develop robust systems.

The proposed methodology is implemented in a constrained complex system of Tamil Nadu Tuticorin Power System (TTPS). The thermal power plant is a complex system with many sub-systems. Constraints are defined for intra class invariants, for inter class invariants and multiplicity constraints. The functional behaviour of the constrained objects is defined and tested with instance values that are validated against the constrained system. Some of the critical situations like cause of alarms, tripping cases of thermal plant are also validated using the OCL based constrained system. The attribute instances of the constrained objects are generated using CSP. Through XML Meta data Interchange (XMI), which is intended to help programmers to exchange data models, a better framework for modeling of any complex system is provided.

2. Background

Modeling of complex systems is an area in which research activities are enormous. The relevance of modeling is that it has a significant role in software engineering as it is the back bone on which the system depends. Object oriented modeling (OOM) helps to map each element of the system as a real world object with characteristics and functions. In real-time complex systems, constraints on characteristics of objects are very crucial and the behaviour of the system depends on these constraints. Hence there is a necessity to represent the constraints in the model.

Many works are done on developing approaches supporting constraint based programming. Bharat Jayaraman *et al*. have developed a programming language and execution environment "Cob" for modeling complex engineering entities [1]. Research on UML diagrams, especially class diagrams includes automatic generation of OCL constraints from UML class diagrams explained by Li Tan *et al*.[2]. Parallel to the release of UML 2.0, a new version of Object Constraint Language has been published. The features of the developed OCL 2.0 at University of Munich are explained by its developers [3][4]. An evaluation of the past, present and future of OCL is explained by Pandey [5].

The explanation on why OCL is used for precise modeling with UML is explained by Duarte *et al*. [6]. The emerging need for the verification techniques to find and notify defects in real life models was justified by Jordi Cabot *et al*.[7]. Markovie *et al*. explain how the OCL annotations will be updated when the underlying class diagram changes [8]. Ricardo *et al*. introduced a new modelling language COMMA, a solver independent platform for modelling constrained objects involving continuous and discrete domains [9]. Ali Hamic *et al.*. have explained the various aspects of syntax and semantics of OCL and the extensions in OCL [10].

Sendall *et al*. have explained enhancing OCL for specifying Pre and Post conditions[11]. Cabot *et al*. have addressed the problem in verification of UML models [12]. Roman has given a survey on Constraint Programming technology and its applications [13]. A report on how to bridge the gap between object oriented software engineering methods and formal verification is presented by Wolfgang *et al*. [14]. Jiang *et al*. proposed the usage of OCL in executable UML[15].

The relevance of constraints in the conceptual models for verification and validation is emphasised by Elita *et al*. [16]. A comparison of the existing approaches for the verification of finite satisfiability in class diagrams is done [17]. Kahina *et al*. have proposed an automatic method of OCL transformation after UML class diagram refactoring [18]. The theoretical and practical views in designing a general purpose constraint solver have also been the area of research for many researchers. A platform "ECLiPSe" for constraint logic programming was also introduced by Wallace *et al*. [19].

An approach that enables the run-time verification of design implementation of constraints is explained by Yoonsik *et al*. [20]. An automatic test case generation for UML object diagrams using Genetic algorithms is proposed by Prasanna *et al*. [21]. The possibilities of transforming OCL formulae to simple rules is also studied [22].

Cadoli *et al*.have explained the finite satisfiability of UML class diagrams using constraint programming [23]. Ali *et al*. proposed a search based OCL constraint solver [24]. The formal verification of UML diagrams through a translation process is verified [25]. Hladik *et al*. have solved a real time allocation problem using constraint programming [26]. Hector *et al*. have developed a model-based approach for testing whether or not an implementation satisfies the constraints imposed by its design model [27]. Imran *et al*. have developed a novel approach for transforming constraints in natural language to OCL constraints [28].

Here, a methodology is proposed for ensuring the correctness of UML class diagrams annotated with OCL constraints by converting to an intermediate form of Constraint satisfaction problem. Validation instances could also be obtained from the solution of CSP which could be used for bounded validation. Hence, the proposed methodology doubles the advantage of facilitating both model verification and model validation.

3. Fault Detection From Object Constrained Models

Software verification is one of the long standing goals of software engineering. The need for correct software specifications is even more relevant in the context of the Model Driven Development (MDD) and Model Driven Architecture (MDA) communities where software models are used to (semi)automatically generate the implementation of the final software system [7]. Hence, the validations and verifications of the model of any system should be great importance since it helps in the creation of a software system that is highly robust in nature.

But, the model by itself cannot be verified. Therefore, it needs to be converted to an executable form to carry out verifications. Source code can be generated through the automatic code generation process, from the verified model. This helps in the coding stage of software development since the validations done on the model will be reflected in the code, thereby reducing extraneous coding to a great extent.

The proposed architecture for validations, verification and fault detection from the UML model of an object constrained complex system by converting into CSP is given in Figure 1.

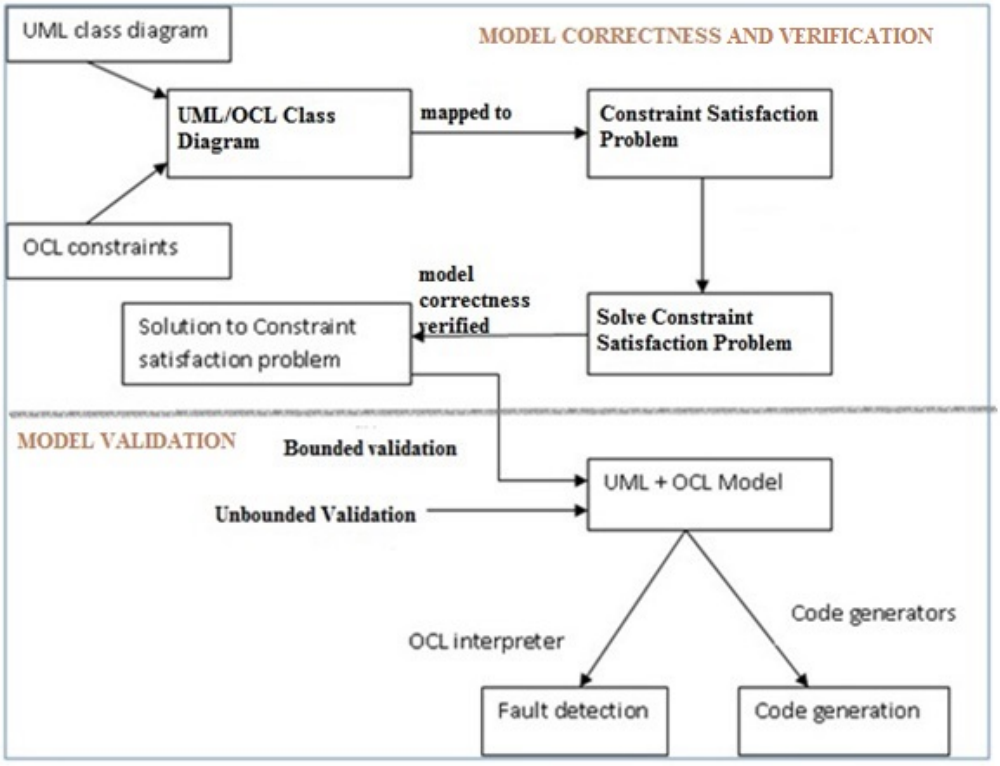

Fig 1: Architecture of a fault detection model using CSP

The proposed architectural diagram addresses four main functionalities.

- Model Verification
- Model Validation
- Fault Detection
- Code Generation

3.1 Model Verification

Assessment of the correctness of the models is a key issue to ensure the quality of any application. A fundamental semantic correctness notion in static models is model satisfiability. The importance of satisfiability comes from the ability to define many other correctness properties, such as liveliness, constraint redundancy and constraint subsumption.

The subtasks involved in Model Verification are:

- Class Diagram Design
- Constraints Specification
- Conversion of Class Diagram annotated with constraints to Constraint Satisfaction Problem (CSP)
- Solve CSP
- Assess the correctness of the model

3.1.1 Class Diagram Design

Class diagrams are the main source to capture the structural information in a system. It represents the modular structure of the system with its static aspects in terms of classes, objects, methods and its relationships. It also helps to perform better analysis on the structure of classes and how they will interact with each other before actually writing any code. Class diagram acts as a blueprint of the system, which also helps for programmers to get an overview of how the application is structured before examining the actual code. This may reduce maintenance effort.

3.1.2 Constraint Specifications

A constraint is a restriction on one or more values of an object oriented system. Though the UML class diagrams are convenient and easy for understanding, from the point of view of automation, these diagrams may moot ambiguity and misinterpretations. Hence, the formal specification language, Object Constraint Language (OCL) is used which helps to represent the structural relationships in an unambiguous manner. OCL is not a programming language and hence it is not possible to write program logic or flow control in OCL [29]. It is purely an expression language, which supplements UML class diagrams in many ways.

- To specify invariants on classes and types in the class model
- To specify type invariant for Stereotypes
- To describe pre- and post conditions on Operations and Methods
- To specify constraints on operations

OCL also helps in automating the process of verification and validation of UML models [30]. The constraints on class diagrams include the intra-class constraints, inter-class constraints, collectively called invariants and multiplicity constraints. OCL expressions are also written to specify constraints on operations of objects in the form of pre and post conditions. Pre-condition specifies the constraint that should be satisfied before the execution of a method and post-condition is the constraint that should be satisfied after the execution of an operation.

The two categories of constraints discussed here are:

- Generic Constraints
- Domain Specific Constraints.

3.1.2.1 Generic constraints

These constraints can be used for any class diagram. The set of generic constraints defined for any class in a class diagram is given in Table 1.

Table 1: Generic Constraints

Constraint	**Definition**
self.classname.size()>0	Class name should not be null
name_for_association= name.size()>0	Role names must be specified
self.generalisation →size()>0	Constraint on the maximum generalization size.
type_for_parameter = ownedParameter→ forAll (n|n.direction<> ParameterDirectionkind::return implies not type.oclIs Undefined())	Data type of the arguments to functions must be defined
self.generalisation→excludes(self)	Self generalisation is not possible

The main advantage of developing generic constraints is that this could be exported as XMI (XML Metadata Interchange) since it provides a platform for the constrained system to be exported to any environment for validating any complex model.

3.1.2.2 Domain Specific Constraints

These are application dependent constraints. For class diagrams, invariants on class attributes and pre and post conditions for methods are specified pertaining to the each element of the system being modeled.

3.1.3 Class Diagram to Constraint Satisfaction Problem

In this work, an attempt is made to convert the class diagram of a complex system annotated with OCL constraints to a Constraint Satisfaction Problem (CSP). Development of a system based on Constraint Programming (CP) includes two phases.

- Problem is formulated as a CSP
- Formulation is implemented in a Constraint Programming Language

A finite domain is considered for the attribute values during the verification process, which is accomplished through the constraints defined in the OCL based constrained system. This way, the constraint solver is able to perform a complete search within the solution space. The main advantage of this conversion is that a pre-defined set of correctness properties of the UML/OCL diagram can be checked using the resulting CSP.

A solution to a CSP is checking whether the model is able to give a finite instance of values such that all constraints are satisfied. A constraint solver is in charge of providing a solution to the CSP. In this work, the constraint solver is the backtracking algorithm which facilitates this checking. Cabot *et al*. have defined correctness properties of models as Finite Satisfiability, lack of constraint subsumption and lack of constraint redundancy [31].

In the proposed method, these model correctness characteristics can be verified from the solution of the CSP. A solution to a CSP is an assignment of values to variables that satisfies all constraints, with each value within the domain of the corresponding variable.

The solutions may be:

- Only one solution
- Many solutions
- No solution

A CSP that does not have solutions is called unfeasible and the corresponding model is incorrect.

Solution to CSP can be found by searching through the possible assignments of values to variable. Here, CSP is solved using the traditional technique, "backtracking", which is explained in Procedure 1. It is implemented by assigning values to variables according to some heuristic and checking whether any constraint is violated after each assignment. If any constraint is violated by a partial solution, the algorithm reconsiders the last assignment, trying a new value in the domain and backtracking to previous variables if there are no more values. This systematic search continues until a solution is found or all possible assignments have been considered. To ensure termination, the search space must be finite, thus, all variable domains must be finite.

```
function BACKTRACKING-SEARCH(csp) return a solution or failure
        return RECURSIVE-BACKTRACKING({} , csp)
function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or
failure
if assignment is complete then return assignment
                var←SELECT-UNASSIGNED
VARIABLE(VARIABLES[csp],assignment, CSP)
                for each value in ORDER-DOMAIN-VALUES(var, assignment, csp)
    do
                if value is consistent with assignment according to
    CONSTRAINTS[csp]
                then
                        add {var=value} to assignment
                        result ← RECURSIVE-BACTRACKING(assignment, csp)
                        if result ≠failure then return result
                        remove {var=value} from assignment
                        return failure
```

Procedure 1: Backtracking algorithm

The results obtained as solution to Constraint Satisfaction Problem can be used to check the model correctness characteristics. If the correctness properties are not satisfied, then the model is unsatifiable and even a single instance could not be brought out that satisfies all the constraints. The results in the solution space reveal that the model is correct and the instance values of the attributes drawn as solution could be used to validate the model.

3.2 *Model Validation*

The overall objective of V&V approaches is to ensure that the system is free from failures and meets its user's expectations. Here model validation means providing instance values to the attributes and functions of objects and validating these instance values against the constraints defined by the OCL based system. Instance values are provided for intra-class attributes, dependent attributes and for the arguments of functions.

Though the solution of CSP can be provided as instance values to the model, the main limitation is that it is a bounded validation. Any model, to ensure the quality, should be subject to unbounded validation also. Here, it is accomplished by providing instance values from the real working system to the model. For any complex real-time system, actual on-site data can be provided to the model for checking faults or critical situations. The validation results reveal the presence or absence of faults, on the basis of the attribute values of parameters.

3.3 *Fault Detection from model*

The main functioning of model validation is to verify whether the model is able to identify those instance values that violates the constraints. Fault handlers or relevant mechanism for handling faults can also be developed, thereby creating robust software.

3.4 *Code generation from model*

The main objective of Model-Driven Architecture itself is the development of the final software system from the model either automatically or semi-automatically. The verified and validated UML class diagram which is annotated with OCL constraints are converted to the final software system by code generators. The reflection of constraints in the code helps to reduce unnecessary coding for verification, which is usually deferred to testing stage of software development.

4. Simulation Results For Fault Detection In TTPS

The system taken for study and implementation is a Thermal Power Plant, TTPS (Tamilnadu Tuticorin Power System) which is a constrained complex system. The Thermal power plant consists of many subsystems, each of which itself is complex.

4.1 *Basics of Thermal Power Plant*

In a conventional thermal power station, a fuel is used to heat water, which gives off steam at high pressure. This in turn drives turbines to generate electricity. At the heart of a power station is a generator, a rotating machine that converts mechanical energy into electrical energy by creating relative motion between a magnetic field and a conductor. The energy source harnessed to turn the generator depends chiefly on which fuels are easily available and on the types of technology used. When coal is used for electricity generation, it is usually pulverised and then burned in a furnace with a boiler. The furnace heat converts boiler water to steam, which is then used to spin turbines which turn generators to produce electricity. The Power plant taken for study , TTPS is a coal-fired thermal power plant.

4.2 *UML Model of the Thermal Power Plant*

From the basic functioning of the thermal power plant, the different classes are identified along with their attributes and functions. All the classes have attributes that are rigid as the model is a complex model with well defined characteristics. The constraints are represented using the formal language , Object Constraint Language (OCL).

The different subsystems of TTPS identified as classes are:

- CoalStorage
- PowerPlant
- Pulveriser
- Boiler
- Furnace
- Drum
- Fan
- ForcedDraft
- InducedDraft
- PrimaryAir
- GasRecirculation
- Turbine
- Condenser
- CoolingTower
- Generator
- Transformer

All these sub-systems have characteristics and functionalities. Table 2 summarizes the attributes and functionalities of four main classes.

Table 2: A few classes of Thermal Power plant with attributes and functions

Class Name	Attributes	Functions
CoalStorage	CalorificValue FixedCarbon Moisture Ash Volatility CoalSize Capacity HGI Amt_of_coal	Add(t) Remove(t)
PowerPlant	Stage Capacity	Startup() Shutdown()
Pulveriser	speed type capacity coalHGI coalMoisture coalFineness rawcoalflow pairflowrate coalinlettemp prairdiffpress outlet_temp diffpr millcurr input_size_coal out_coal_surface_moisture out_coal_size out_coal_surface_area grindingtype	Drying() Grinding() Circulation() Classification() Startup() Shutdown()
Boiler	capacity outlet_pr temp RH_Inlet_pr RH_Outlet_pr RH_Inlet_temp RH_Outlet_temp Drum_pr Feedwater_temp Coal_load Oil_load	Startup() Shutdown() Preheating() Reheating()

A snapshot of the class diagram created in Eclipse IDE is given in Figure 2.

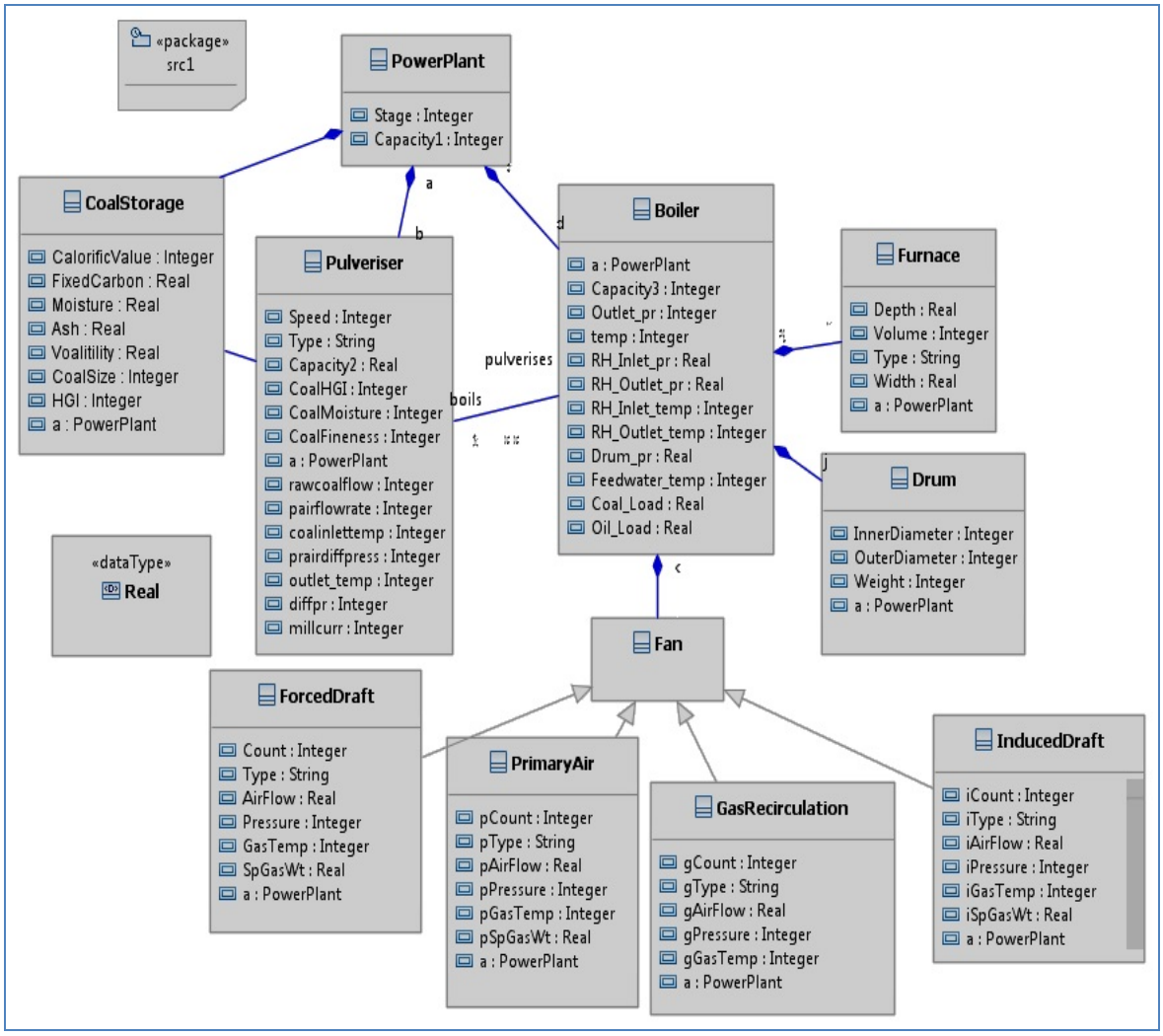

Fig 2:: Class Diagram of Thermal Power Plant

4.3 *Constraints Specifications*

Constraints are defined as invariants for the attributes of the classes and also as preconditions and postconditions for functions. Preconditions / Postconditions are constraints that specify the applicability and effect of an operation without stating an algorithm or implementation. These are constraints that must be true just prior to the execution of an operation and after the execution of the function. A few domain specific constraints pertaining to the subsystems “CoalStorage”, “Pulveriser” , “Boiler”, “PowerPlant”, “Furnace”, “Drum”, “ForcedDraft”, “InducedDraft”, “PrimaryAir” and “GasRecirculation” are presented.

Table 3: OCL constraints of TTPS

context CoalStorage
inv: if a.Stage = 3then
self.Ash=40.4
else
self.Ash=19
endif
	context CoalStorage
	inv : if a.Stage = 3 then
	self.Volatility=27.6
	else
	self.Volatility=33
	endif

context CoalStorage
inv : if a.Stage = 3 then
self.CalorificValue=3855
else
self.CalorificValue=5950
endif
	context PowerPlant
	inv: stvalue: Stage<=3
	inv: if Stage=2 then
	Capacity1 = 210
	else
	Capacity1=420
	endif

context Boiler
inv : if a.Stage = 3 then
self.Capacity3=690
else
self.Capacity3=700
endif
	context Boiler
	inv : if a.Stage = 3 then
	Outlet_pr=155
	else
	Outlet_pr=137
	endif

context Boiler
inv : if a.Stage = 3 then
Oil_Load=52
else
Oil_Load=51.2
endif
	context ForcedDraft
	inv: Count=2
	context ForcedDraft
	inv: GasTemp=50
	context ForcedDraft
	inv: SpGasWt = 1.07

context InducedDraft inv:
if a.Stage=3 then
iGasTemp=157
else
iGasTemp=140
endif
	context InducedDraft inv:
	if a.Stage=3 then
	iSpGasWt=0.80
	else
	iSpGasWt=0.83
	endif

context PrimaryAir inv: pGasTemp=50	context PrimaryAir inv: pSpGasWt=1.07

context GasRecirculation inv:
if a.Stage=1 or a.Stage=2 then
gCount=2
else
gCount>0
endif
	context GasRecirculation inv:
	if a.Stage=1 or a.Stage=2 then
	gAirFlow=113.4
	else
	gAirFlow<>0
	endif

A few examples of Pre-conditions and Post-conditions of TTPS system are presented.

Table 4: Pre-conditions and post-conditions of TTPS

context CoalStorage::add(t:Float) pre : t>0.0 capacity>=t post: wt_of_coal=wt_of_coal@pre+t	context CoalStorage::remove(t:Float) pre : t>0.0 wt_of_coal>t post: wt_of_coal=wt_of_coal@pre-t
context Pulveriser::classification() pre : none post: out_coal_size<=8	context Pulveriser::drying() pre : coalinlettemp<=300 post: out_coal_surface_area> out_coal_surface_area@pre out_coal_surface_moisture<out_coal_surface_moisture@pre

Startup() and Shutdown() are two operations of all classes. They are used to ensure the constraints on the values of the parameters before the start-up of that process as precondition. Shutdown() also ensures the constraints before and after shutdown of that subsystem as pre and post conditions.

4.4 *TTPS to Constraint Satisfaction Problem*

To achieve the formal verification of the UML model, it needs to be converted to executable code. Here, to analyse the correctness of the class diagram of TTPS with OCL constraints, it is converted to CSP. For each element of the class diagram, a set of variables, domains and constraints are defined as <attribute, domain, constraint>. For instance, the Pulveriser class is mapped to CSP as given below.

Pulveriser <rawcoalflow,{positive real number},rawcoalflow<=45>
<outlet_temp,{positive real number}, outlet_temp<=100>
<diffpr,{positive real number}, diffpr<=500>
<millcurr,{positive real number}, millcurr<=60>
<pairflowrate,{positive real number}, pairflowrate<=75 >

UMLtoCSP proposed by Cabot *et al*. translates UML class diagrams annotated with OCL constraints to CSP [Cabot, 2008].

The automatic computation of domains from constraints is a complex problem. The backtracking algorithm is used to find solutions to the problem and the correctness of the model is verified from the solution of CSP.

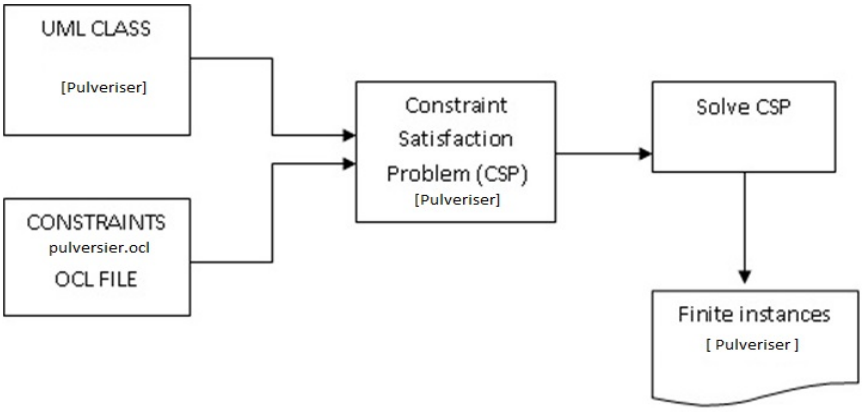

Fig 3: Model Verification Process

Table 5: CSP solutions

contr.. / instances	RCF RCF<=45	OLTEMP OLTEMP<=100	CINTEMP CINTEMP<=300	MS	LCR	LCS
Instance1	40	50	150	✓		
Instance2	40	55	310		✓	✓
Instance3	35	60	290	✓		
Instance4	46	90	290		✓	✓
Instance5	30	150	350		✓	✓

The solutions of CSP summarized in Table 5 are a few instances from which the model correctness characteristics can be verified.

RCF : rawcoalflow
OLTEMP : outlet_temp
CINTEMP : coalinlettemp
MS : Model Satisfiability
LCR : Lack of Constraint Redundancy
LCS : Lack of Constraint Subsumption

Verification results reveal that the model can provide finite instances in a manner that all the constraints are satisfied and also that there is no constraint contradiction and constraint subsumption.

4.5 *Model Validation*

The next phase is model validation, where the degree of association between the model and the real system is judged. Model validation helps to establish the quality of the model through its representational accuracy. The results of validation reveal the correctness of the model coded.

In the real system of TTPS in C-DAC, fault is detected based on a rule base which consists of all the constraints. Manual intervention is necessary on reception of an "alarm" which indicates a fault. The TTPS system is maintained without faults by mathematical modeling of the coal mill [32]. In this work, the UML class diagram annotated with OCL constraints is validated by providing instance values to the model. The model is validated against the OCL based constrained system.

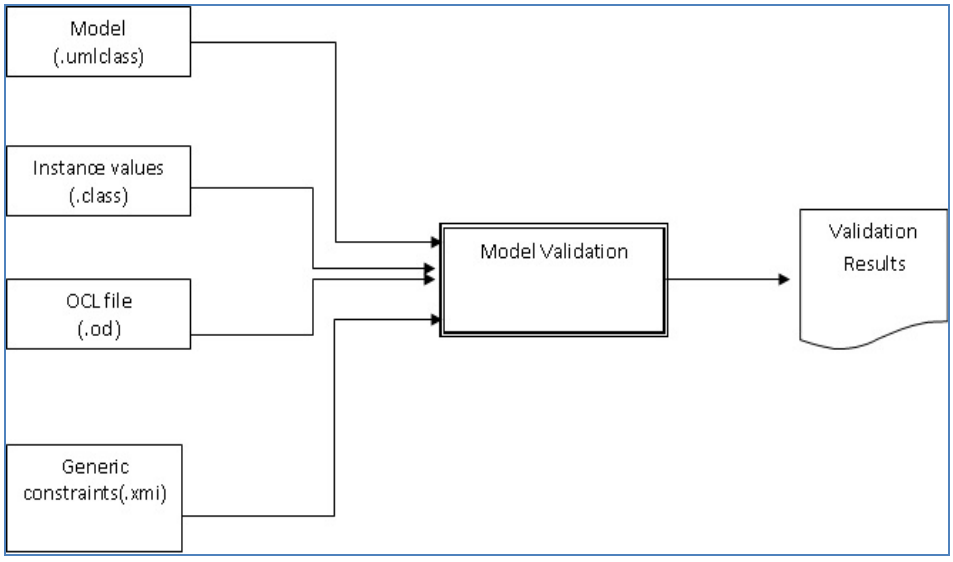

Fig 4: Model Validation Process

Model Validation is performed using Dresden OCL Toolkit [Dresden]. It also aids code generation process [33]. The inputs for validation are:

- The model is the UML class diagram of TTPS
- Instance values are provided through the ModelInstanceProviderClass, and the values are the solution results of CSP – Bounded Validation
- The constraint file with all invariants, pre-conditions and post-conditions on all subsystems
- The set of generic constraints exported as .xmi

When these values are provided for model validation, the OCL interpreter evaluates the instance values against the constraints and produces the results. The model is also validated with real instance values obtained from the thermal power plant site for unbounded validation.

4.6 *Fault Detection*

Model Validation facilitates two processes:

- Fault Detection from model
- Code generation from model

Figure 5 shows the fault detection process when the pulveriser sub-system is validated.

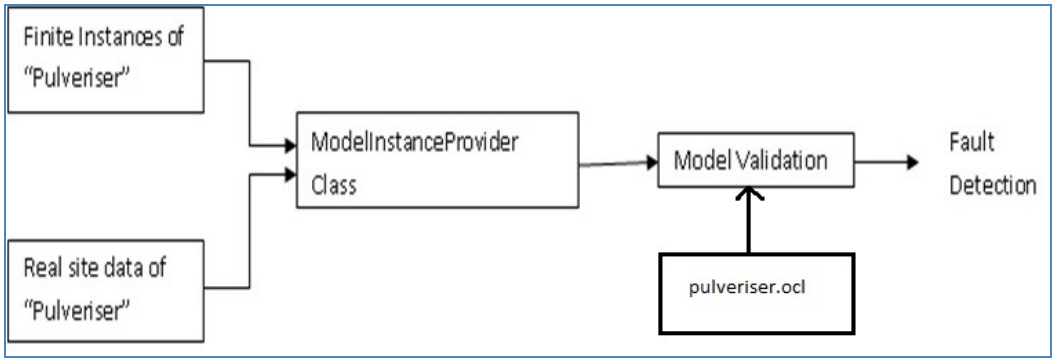

Fig 5: Model Validation of Pulveriser

A few results obtained when the model is validated is summarised in Table 6.

Table 6: Fault Detection Results

Object	Instance values	Fault/Correct
Boiler	[Capacity=700 OutletPressure=137 Temperature=540 RH-Inlet-Pressure=26.35 RH-Outlet-Pressure=24.5]	Correct
Pulveriser	[Speed=52 Capacity=33.8 CoalFineness=0.7 CoalMoisture=0.1 Rawcoalflow=40 Pairflowrate=70 Coalinlettemp=250 Prairdiffpress=150 outlet_temp=90 diffpr=450]	Correct
Coal Storage	[CalorificValue=5950 FixedCarbon=40.5 Moisture=7.5 Ash=21 Volatility=33 CoalSize=20]	Fault/Alarm [Constraint violated : Ash=19]

4.7 *Code generation*

Model driven engineering has the major advantage of automatic or semiautomatic code generation from the model. Constrained system modeling includes the structural and behavioural modeling along with the constraints. If the constraints are reflected in the code generated from the model, then extraneous coding can be avoided to a great extent.

From the UML Class Diagram and the constraint file pertaining to the selected model, a code is generated automatically using Dresden OCL toolkit [34][35]. The constraints defined in the OCL file, are reflected in the ultimate code that is generated.

```
package src1.constraints;

    if (((Object) aClass.der).equals(new
Integer(3)))
      {
      ifExpResult1=
((Object)aClass.Capacity3).equals(new
Integer(690));
        }
      else
       {
       ifExpResult1=
((Object) aClass.Capacity3).equals(new
Integer(700));
        }
    if (!ifExpResult1)
       {

       String msg = "Error: Constraint '" +
undefined + "' ("inv : if der = 3 then
self.Capacity3= 690 else self.Capacity3=
700 endif + ") was violated for Object " +
aClass.toString() + "";
       throw new RuntimeException(msg);
        }
```

Fig 6: Sample source code generated with constraints from class diagram

Many control systems like the steam temperature control system of the boiler component of thermal power plant have been developed using genetic algorithms [36]. Many works have been carried out in coal-mill modeling. A six segment coal-model that covers the whole milling process from mill start-up to mill-shut-down is studied [37]. Modeling coal-mill by machine learning with on-site data was also carried out [38]. In all these works, mathematical modeling of the milling process is done and is optimised at an observable level. Genetic Algorithms were used to find an optimal value to the parameters K1..K17 used in the mathematical modeling of a coal-mill in the real thermal power plant site [39]. Strong mathematical background is very much essential for modelling a coal-mill using any of the above mentioned methodologies. Moreover, direct translation to the source code is not possible which requires the developer to explicitly translate all the constraints in the model to the source code. But, using the proposed approach, direct translation to the code is possible with constraints reflected in it, thereby reducing extraneous coding to a great extend.

5. Conclusion

In this work, a static model, of a constrained complex system is created, and the model correctness characteristics are verified on the model by converting it to executable code. The verified model is validated against instance values both by unbounded and bounded validation. Fault could be detected from the model, when the instance values violated the constraints. The fault detection routines and error handling procedures written as constraints were reflected in the code generated from the model. The methodology is tested on TTPS and the results of fault detection were in accordance with the real scenario.

But, the proposed methodology can be employed only to detect failures from the model of thermal power plant system and failure prediction is not addressed, which is highly relevant for a constrained complex system. Hence the future direction is to incorporate into this model and failure prediction system with which failures can be predicted.

References

[1] Jayaraman, B. and Tambay, P. “Constrained objects for modeling complex structures”, OOPSLA 2000,pp.71–76.
[2] Tan, Li, Zongyuan Yang, and Jinkui Xie. "OCL constraints automatic generation for UML class diagram." 2010 IEEE International Conference on Software Engineering and Service Sciences. IEEE, 2010.
[3] Hussmann, Heinrich, and Steffen Zschaler. "The object constraint language for UML 2.0–overview and assessment." Upgrade Journal vol.5no.2 2004.
[4] Hennicker, Rolf, Heinrich Hussmann, and Michel Bidoit. "On the precise meaning of OCL constraints." Object Modeling with the OCL. Springer Berlin Heidelberg, 2002. 69-84.
[5] Pandey, R. K. "Object constraint language (OCL): past, present and future." ACM SIGSOFT Software Engineering Notes vol.36.no.1 2011: pp.1-4.
[6] Duarte, R., J. Júnior, and A. Mota. "Precise modeling with UML: Why OCL?." Submitted to the Workshop of Formal Methods. 2003
[7] Cabot, Jordi, Robert Claris, and Daniel Riera. "Verification of UML/OCL class diagrams using constraint programming." Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on. IEEE, 2008.
[8] S. Markovic and T. Baar, ”Refactoring OCL annotated UML class diagrams”, Software and System Modeling vol.7 no.1 ,2008, pp 25–47.
[9] Soto, Ricardo, and Laurent Granvilliers. "The design of comma: An extensible framework for mapping constrained objects to native solver models." 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007). Vol. 1. IEEE, 2007.
[10] Hamie, Ali, et al. "Reflections on the object constraint language." International Conference on the Unified Modeling Language. Springer Berlin Heidelberg, 1998.
[11] Sendall, Shane, and Alfred Strohmeier. "Enhancing OCL for specifying pre-and postconditions." UML 2.0-The Future of the UML Object Constraint Language (OCL). Workshop of UML 2000-The Unified Modeling Language: Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000.. No. LGL-CONF-2000-005. 2000.
[12] Cabot, J., Clarisó, R. and Riera, D. “Verification of UML/OCL class diagrams using constraint programming”,ICST Workshops 2008, pp.73–80
[13] Barták, Roman. "Constraint) Programming!." Pursuit of the Holy Grail Proceedings of WDS99 (invited lecture), Prague (1999).
[14] Ahrendt, Wolfgang, et al. "The Approach: Integrating Object Oriented Design and Formal Verification." European Workshop on Logics in Artificial Intelligence. Springer Berlin Heidelberg, 2000.
[15] Jiang, Ke, Lei Zhang, and Shigeru Miyake. Using OCL in executable UML. Technische Universität Berlin, 2008.
[16] Miliauskaitė, Elita, and Lina Nemuraitė. "Representation of integrity constraints in conceptual models." Information technology and control vol 34no.4, 2015.
[17] Bastos, Paulo, and Pedro Ramos. "finite satisfiability verification in UML class diagrams-a comparative study." IADIS International Journal on Computer Science & Information Systems vol.8.no.1 2013.
[18] Hassam, Kahina, Salah Sadou, and Régis Fleurquin. "Adapting OCL constraints after a refactoring of their model using an MDE process." 9th edition of the BElgian-NEtherlands software eVOLution seminar (BENEVOL 2010). 2010.
[19] Wallace, Mark, Stefano Novello, and Joachim Schimpf. "ECLiPSe: A platform for constraint logic programming." ICL Systems Journal vol.12no.1 1997: pp.159-200.
[20] Cheon, Yoonsik, et al. "Checking design constraints at run-time using OCL and AspectJ." 2009.
[21] Prasanna, M., and K. R. Chandran. "Automatic test case generation for UML object diagrams using genetic algorithm." Int. J. Advance. Soft Comput. Appl vol.1.no.1 2009:pp. 19-32.
[22] Giese, Martin, and Daniel Larsson. "Simplifying transformations of OCL constraints." International Conference on Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, 2005.
[23] Cadoli, Marco, et al. "Finite satisfiability of UML class diagrams by Constraint Programming." CSP Techniques with Immediate Application (CSPIA) 2 2004.
[24] Ali, Shaukat, et al. "A search-based OCL constraint solver for model-based test data generation." 2011 11th International Conference on Quality Software. IEEE, 2011.
[25] Smith, Jeffrey E., Mieczyslaw M. Kokar, and Kenneth Baclawski. "Formal Verification of UML Diagrams: A First Step Towards Code Generation." pUML. 2001
[26] Hladik, Pierre-Emmanuel, et al. "Solving a real-time allocation problem with constraint programming." Journal of Systems and Software vol.81no.1 2008: 132-149.
[27] Chavez, Hector M., et al. "An Approach to Checking Consistency between UML Class Model and Its Java Implementation." IEEE Transactions on Software Engineering vol.42no.4 2016: 322-344.
[28] Bajwa, Imran Sarwar, Mark Lee, and Behzad Bordbar. "Translating natural language constraints to OCL." Journal of King Saud University-Computer and Information Sciences vol.24no.2 2012: 117-128.
[29] UML 2.0 OCL specification
[30] Pandey, R. K. "Object constraint language (OCL): past, present and future." ACM SIGSOFT Software Engineering Notes vol.36.no.1 2011: pp.1-4.
[31] Cabot, J., Clarisó, R. and Riera, D. “Verification of UML/OCL class diagrams using constraint programming”,ICST Workshops 2008, pp.73–80.
[32] Singh, B. Raja, et al. "Real Time Pulverised Coal Flow Soft Sensor For Thermal Power Plants Using Evolutionary Computation Techniques." ICTACT Journal on Soft Computing vol.5.no2 2015,pp.911-916.
[33] Demuth, Birgit. "The Dresden OCL toolkit and its role in Information Systems development." Proc. of the 13th International Conference on Information Systems Development (ISD’2004). Vol. 7. 2004
[34] Wilke, Claas, and Ronny Brandt. "An Introduction into Dresden OCL2 for Eclipse." 2008.
[35] Demuth, Birgit, and Claas Wilke. "Model and object verification by using Dresden OCL." Proceedings of the Russian-German Workshop Innovation Information Technologies: Theory and Practice, Ufa, Russia. 2009
[36] Valsalam, S.R., Anish, S. and Singh, B.R. “Boiler modeling and optimal control of steam temperature in thermal power plants”, Journal of Energy and Power Engineering, Vol. 5,No 8,2011, pp.677–684.
[37] Wei, J-L. et al. “Development of a multi segment coal mill model using an evolutionary computation technique”, IEEE Transactions On Energy Conversion, September2007, Vol. 22, No. 3, pp.718–727.
[38] Zhang, Y.G., Wu, Q.H., Wang, J., Oluwande, G., Matts, D. and Zhou, X.X. “Coal mill modeling by machine learning based on onsite measurements”, IEEE Transactions on Energy Conversion, December, vol. 17, no. 4, pp.549–555.
[39] Singh, B. Raja, et al. "Real Time Pulverised Coal Flow Soft Sensor For Thermal Power Plants Using Evolutionary Computation Techniques." ICTACT Journal on Soft Computing vol.5.no2 2015,pp.911-916.