Implementing object oriented design methodology to develop a testing system.

Danendra Singh (Electrical engineering department, Delhi Technological University, India)
Leonardo M. Reyneri (Electronics and telecommunications department, Politecnico di Torino, Italy)

Abstract--Object oriented designing (OOD) is a widely used development strategy based on the concept that systems should be built from a collection of reusable components called objects. Instead of using structures that separates functionality and data, objects bounds both. It guarantees that the system will enjoy a longer life while having far smaller maintenance costs and tends to model the real world in a more complete fashion than do traditional methods. Unified modelling Language (UML), based on OOD, is implemented using Visual Paradigm, a visual design tool. The system is divided into actors, use cases and class diagrams having interactions with each other. Schematic files and simulations that are done in a third party software can be uploaded into the documentation of Class diagrams in UML.

Index Terms-UML, Class Diagram, Actors, Use Cases, Solar cell.

1 INTRODUCTION

Object-oriented design (OOD) is a programming paradigm that began in the late 60's with an increase in the complexity of designs. The idea behind the approach was to build software systems by modeling them based on the real-world objects that they were trying to represent. One of the reasons that OOD has gained wide popularity is due to its systematic approach. The system developed by implementing OOD is easier to maintain, debug and modify.

OOD involves breaking down a system into subsystems and then into elementary objects or blocks. Each of these items are closely associated with their data, functions, interfaces with other objects, and their characteristics.

Unified Modeling Language (UML) is a general-purpose modeling language which is based on the concept of OOD. It was created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software during 1994–95. In 1997 it was adopted as a standard by the Object Management Group (OMG), and has been managed by this organization ever since. UML helps to specify, model, visualize, develop and document an under construction project.

It is interesting to note that that OOD is also similar to the Yourdon De Marco approach to Structured Analysis which is intended to be applied to the design of both HW and SW systems. More recently also UML evolved into a dialect (SysML) which is intended for the design of HW systems

This paper describes an academic approach to Object Oriented Engineering (OOE), also referred to as Model Based Design, and it shows how the native UML language can be effectively used to design hybrid HW/SW systems.

1.1 UML language in brief

A diagram in UML is divided into two categories: Structure diagrams and Behavioral diagrams. Structure diagrams represent the structural formation of a system. It consists the components involved in the modeling. Behavioral diagrams define how the system is going to behave and interact. It defines the functionality of the model.

Visual Paradigm is a software that is used to model UML diagrams. It is an UML design tool and UML CASE tool designed to aid system development. Visual Paradigm supports key industry modeling languages and standards. It offers complete tool-chest that organizations need for capturing the requirements, planning software and testing, class modeling, data modeling, documentation, reporting etc.

The modeling of the system involves defining the use case diagrams containing the use cases and the actors interacting with these use cases. Then the various classes and their association with each other are defined in class diagrams. The classes may also contain subclasses. If there are classes containing softwares or commercial components or mechanical elements, etc. then special stereotypes are given to those classes.

Documentation is an important step while modeling in UML. All the necessary data required to define a class is included in the class. This makes report generation an easy task. Apart from this, documentation helps a lot if some other member of the design team has to use that class or a similar class in his project. Additionally, if a new member joins the project and has to make some modifications in a

- *Danendra Singh is currently pursuing Bachelors degree program in Electrical engineering at Delhi Technological University, New Delhi, India. Phone- +919717278586. E-mail: danendrasingh@dtu.ac.in*
- *Leonardo M. Reyneri is working as a professor in Electronics and telecommunications department at Politecnico di Torino, Italy. Phone- +390110904038. E-mail: leonardo.reyneri@polito.it*

class, appropriate "comments" can be added to the original class.

Using OOD for developing the testing system assisted in time saving and helped in easier debugging of the system. The schematics of the various components (developed using Mentor Graphics tool) were directly uploaded into the Visual Paradigm project. Simulation results from Mentor Graphics was also incorporated inside the documentation of classes.

Another remarkable point is that Visual Paradigm allows you to automatically encapsulate the code embedded into Software classes and produce C++ code for a Microcontroller's Integrated development environment(IDE) like IAR Embedded workbench. This organizes all the key elements of a design (e.g. schematics, C code, mechanical drawings, simulation patterns, documentation) in one place for easier accessibility.

2 Description of Use case diagram

A use case diagram represents the interaction of user with the system. It shows the relationship between the user and the different use cases in which the user is involved.
The Fig. 1. Shows an example use case diagram to demonstrate the concept of OOD.

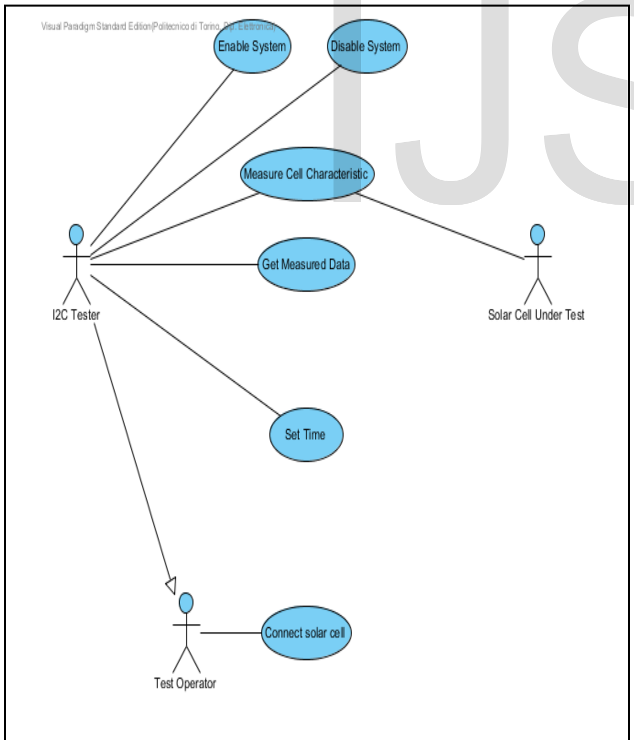

Fig.1. Use case diagram of a demo system

2.1 Actors

These define the human users, external hardware, or other subjects that interacts with the system under consideration via a set of well-defined functions which are called "use cases".

For Example in Fig. 1. the actors are:

Name	Documentation
Test Operator	Test operator could be a Mechanical / Electrical Operator which is responsible for making wired connections of the system.
Solar Cell Under Test	This is the Test object. It interacts with the Current Sensor and Voltage Source.
I2C Tester	The I2C Tester interacts with the test equipment by means of Basic Protocol via an I2C and Logic Supply Connector located on the back of the test equipment. By means of the Basic Protocol , the I2C Tester can instruct the test equipment to perform a set of actions onto the device under test. The I2C Tester can either be a human which uses its own I2C User Interface, or a Main Controller capable of performing a number of complex and highly structured test onto the device under test.

Table 1. Actors for the demo system

2.2 Use Cases

The functionalities of a system written in an organized fashion defines the use cases of the system. They lists the actions or events defining the interactions between an actor and the subsystem.
The use cases in Fig. 1. are:

2.1.1 Connect solar cell

The Test Operator connects physically the Solar Cell Under Test with the system via Connectors.

2.1.2 Disable System

The I2C Tester switches off a safety switch to cut off the connections of Solar Cell Under Test with the system.

2.1.3 Enable System

The I2C Tester switches on a safety switch to make connections of the Solar Cell Under Test with the system.

2.1.4 Set Time

The I2C Tester send a command SET_TIME to the test equipment by using Write Data to set the Time.

2.1.5 Get Measured Data

Retrieves last measured data (from Measure Cell Characteristic).

2.1.6 Measure Cell Characteristic

Start and executes characterization of Solar Cell Under Test.

3 Description of Class diagram

Class diagrams are the basis of object-oriented analysis and design. They show the classes of the system, their inter-relationships, the various operations and attributes of the classes. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modelling and detailed design modelling.
The description/documentation of class diagrams for Fig. 2. Are given as follows:

This contains all the Software and Hardware blocks required for the Subsystem.
It consists of a Voltage Source which takes in a 5V DC input from I2C and Logic Supply Connector and converts it into 3.3V DC which powers the MCU and Sense.
It also has a Programmable Resistor whose resistance can be varied by changing the potential on V_IN(). It is a current sinking device connected to the Solar Cell Under Test and varies the current flowing through it.
Alongwith this, there is a Programmable Voltage Source which generates an output voltage on pin VOUT() proportional to the input voltage on pin V_IN(). This becomes active when there is no light falling on the Solar cell and it is sinking current.
The Programmable Resistor and Programmable Voltage Source are connected to Solar Cell Under Test by an electrically controlled GQ Relay AGQ260A03.
Sense consists of current and voltage sensors necessary for measuring the I-V characteristics of Solar Cell Under Test.

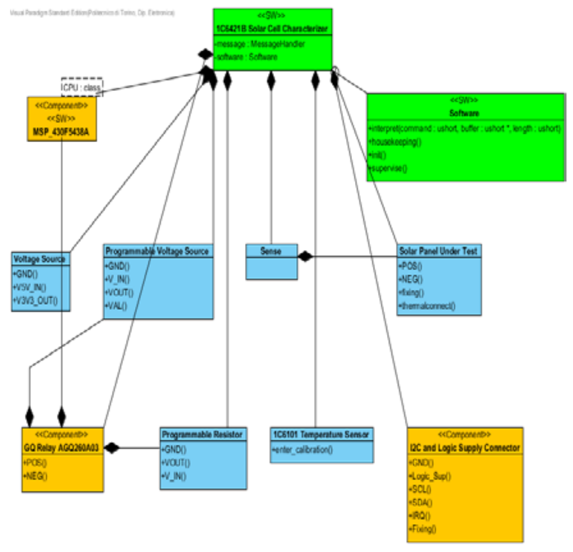

Fig. 2. Class diagram of a demo system

3.1 Sense

Contains a voltage and current sensor.

- The Current sensor is used to measure the current sinked from the IP_POS() pin. The value of hall effect voltage corresponding to the sinked current is available at I_sense() pin.
- The voltage sensor measures voltage across the test object It sinks a small value of current from the test object (eg. solar cell).

3.1.2 Internal Structure

This section describes the internal structure and composition of the sensor Sense, which is composed of an OPAMP plus a current sensor.

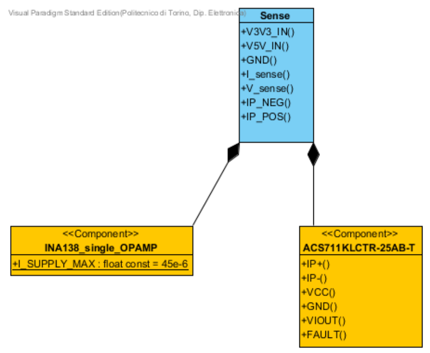

Fig. 3. Internal Structure of Class 'Sense'.

3.2 Programmable Resistor

It acts as a variable resistor (current sinking device), controlled by a DC voltage(supplied by V_IN() pin).
The current flowing though this device can be controlled by the input voltage (supplied to V_IN() pin). This can be attributed to a change in internal resistance.

3.2.1 Operations

Signature	Documentation
GND()	Ground Terminal.
VOUT()	This terminal accepts a variable voltage(0-40V) from the device which sources a current. Can only sink current up to 1A to avoid damage. Cannot source current. Current entering this pin depends on the voltage on pin V_IN(). The relationship between V(V_IN()) and I(VOUT()) is not accurate but it is guaranteed that I(VOUT()) >= 1A when V(V_IN()) = 3.3V.
V_IN()	Control input. The DC voltage between this pin and GND() controls the current through the Programmable Resistor. Minimum voltage is 0V. With minimum voltage, the current through Programmable

	Resistor is zero (namely, less than 10 µA). Maximum voltage is 5V.

Table 2. Operations in the class 'Programmable Resistor'.

3.2.2 Internal Structure

This section describes the internal structure, the design, the simulation and testing of the Programmable Resistor, which is composed of a power MOS plus a power resistor.

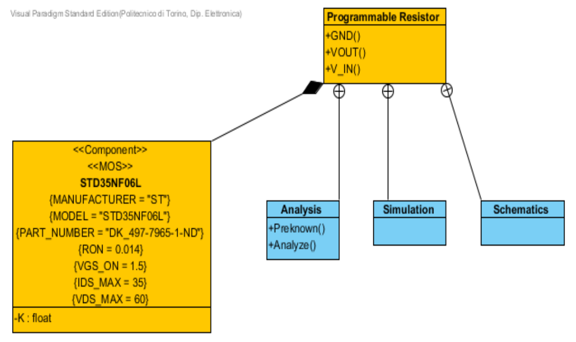

Fig. 4(a). Internal Structure of Class 'Programmable Resistor'

3.2.3 Schematics

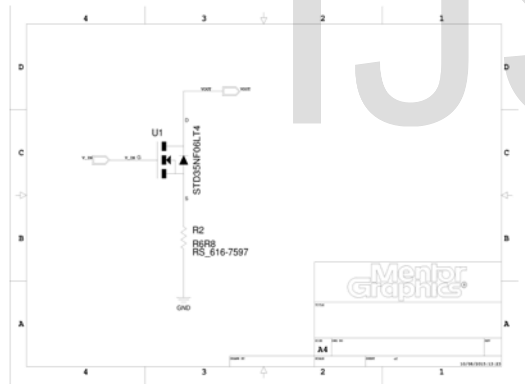

Fig. 4 (b). Schematics incorporated in the documentation of Class 'Programmable Resistor'

3.2.4 Simulation

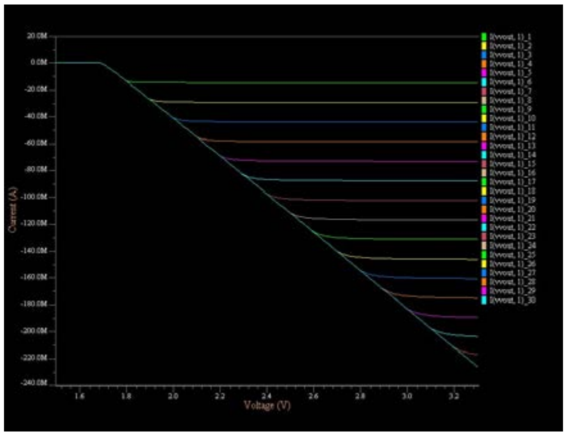

Fig. 5. Simulation results incorporated in the documentation of Class 'Programmable Resistor'

3.3 Programmable Voltage Source

It is a programmable DC voltage source. It generates an output voltage on pin VOUT() proportional to the input voltage on pin V_IN().
The source can generate voltage up to the supply voltage VAL() sourcing up to 250mA. Receives a DC voltage from high impedance V_IN() pin and boosts the voltage by a factor of 1.5. This boosted voltage is available at VOUT() pin.
Output voltage is:
V(VOUT()) = 1.5 * V(V_IN()).

3.3.1 Operations

Signature	Documentation
GND()	Ground Terminal.
V_IN()	Supplies a DC voltage to Programmable Voltage Source.
VOUT()	Output pin. Source a voltage variable between 0V to VAL() proportional to voltage on pin V_IN(). Can only source current up to 250mA. Cannot sink current.
VAL()	Positive supply input; max current is 1A. This Controls the voltage value at VOUT().

Table 3. Operations in the class 'Programmable Voltage Source'.

3.3.2 Internal Structure

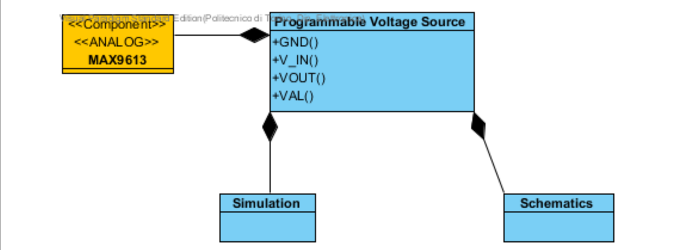

Fig. 6. Internal Structure of Class 'Programmable Voltage Source

3.3.3 Schematics

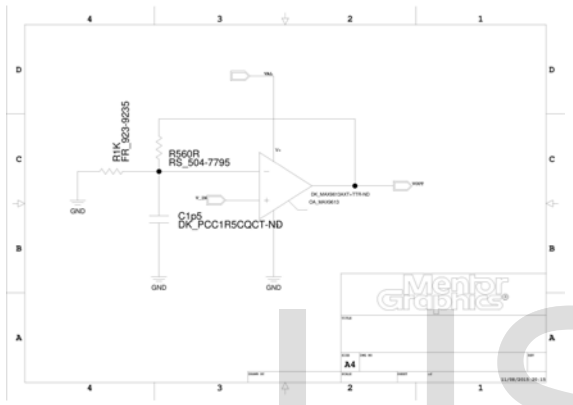

Fig. 7. Schematics incorporated in the documentation of Class 'Programmable Voltage Source'.

3.4 Voltage Source

Supplies a Constant 3.3V output through V3V3_OUT() pin.
Takes in 5V DC input through V5V_IN().
Max output current: 800mA.
Max Input voltage: 20V.
Load Regulation: 0.4%.

3.4.1 Operations

Signature	Documentation
GND()	Ground terminal.
V5V_IN()	5V +/- 5% supply input; max current is 1A.
V3V3_OUT()	Constant voltage output; 3.3V +/- 0.1V; max 800mA V5V_IN() is stepped down to 3.3V.

Table 4. Operations in the class 'Voltage Source'.

3.4.2 Internal Structure

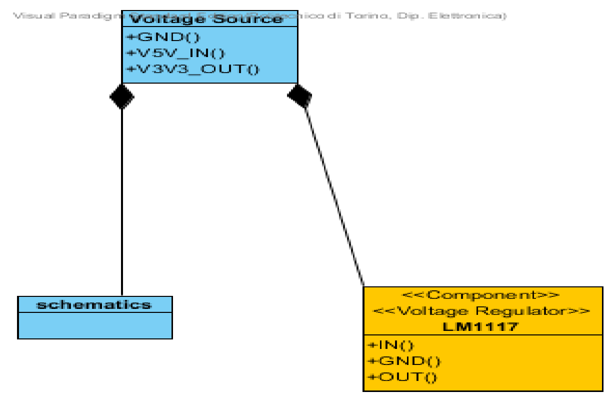

Fig. 8. Internal Structure of Class 'Voltage Source'.significance of the figure in the caption.

3.5 I2C and Logic Supply Connector

External Interface Connector of ControlBoard.
I2C and Logic Supply Connector has following pins
1. GND()
2. Logic_Sup()
3. SCL()
4. SDA()
5. IRQ()

3.5.1 Operations

Signature	Documentation
GND()	Logic GND pin
Logic_Sup()	Logic Supply Pin, 5V, max current 1A
SCL()	I2C clock pin
SDA()	I2C data pin
IRQ()	Interrupt request for I2C

Table 5. Operations in the class 'I2C and Logic Supply Connector'.

3.6 Software

Contains the Software required for measuring the solar characteristics. This contains set of commands that controls the microcontroller to:

- Measure the voltage across the solar cell using ClassSense.
- Measure the current sourced by the solar cell using ClassSense.
- Output the measured current, voltage and time stamp the user using I2C and Logic Supply Connector.
- It also controls whether the solar cell will be sourcing current(when light is falling on it) or it will be sinking current (when there is no light falling on its surface).

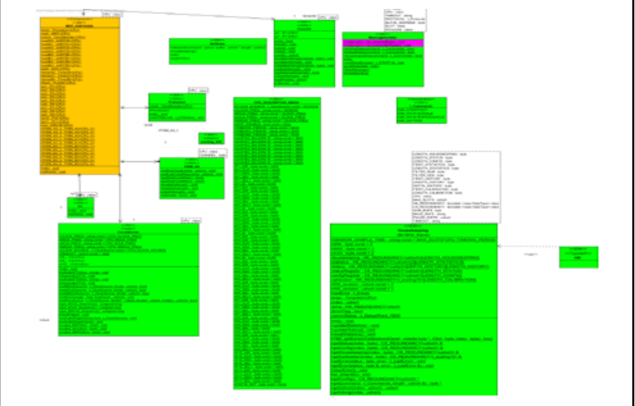

Fig. 9. Structure of Class 'Software'.

3.7 Solar Panel Under Test

This is the Test object. It can also have the reverse protection diode.
It interacts with the Current Sensor and Voltage Source to Measure Cell Characteristic

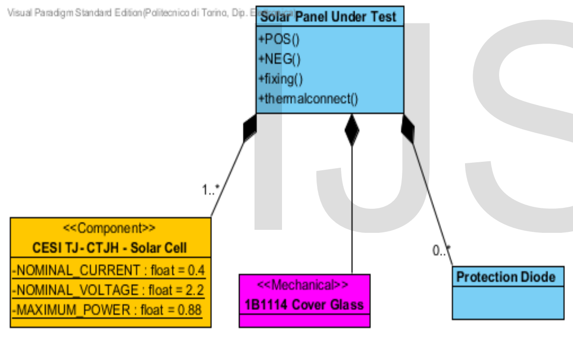
Visual Paradigm Standard Edition(Politecnico di Torino, Dip. E
Solar Panel Under Test
+POS()
+NEG()
+fixing()
+thermalconnect()
1..*
0..*
<<Component>>
CESI TJ-CTJH - Solar Cell
-NOMINAL_CURRENT : float = 0.4
-NOMINAL_VOLTAGE : float = 2.2
-MAXIMUM_POWER : float = 0.88
<<Mechanical>>
1B1114 Cover Glass
Protection Diode

Fig. 10. Structure of Class 'Solar Panel Under Test'.

3.7.1 Operations

Signature	Documentation
POS()	Positive terminal connection for the Solar Panel Under Test.
NEG()	Negative terminal connection for the Solar Panel Under Test.

Table 6. Operations in the class 'Solar Panel Under Test'.

3.8 GQ Relay AGQ260A03

High Sensitivity DPDT switching relay. Used to switch the connection of Programmable Voltage Source or Programmable resistor with Solar Cell Under Test.

3.8.1 Operations

Signature	Documentation
POS()	Connects the Programmable Resistor with Solar Panel Under Test when this control port is high.
NEG()	Connects the Programmable Voltage Source with Solar Panel Under Test when this control port is high.

Table 7. Operations in the class 'GQ Relay AGQ260A03'.

4 Conclusion

The Study of OOD with the help of a testing System example shows that a complex system can be modelled, visualized, modified, presented and even documented very easily by using OOD.
The division of various parts of the subsystem in classes helped in better organization and reusability of the parts. Any new user now takes a lesser time to understand a system as it would have taken without the implementation of OOD. Whenever a project report is required, UML (a tool for OOD) can easily generate it from the documentation of the various classes.
OOD helps in reducing the software complexities of the system. UML links the software coding of a class to an IDE. Hence now, only a small portion of the entire code is required in different global classes which can be combined to generate the main code in the IDE by accessing the different classes in the UML.
This proves that OOD is very effective for system modelling and designing. Improvements in the modelling platforms will encourage more and more people to use it and hence boost their productivity while decreasing their time input.
As next step of study we intend to implement OOD for realizing a complete system for modular nanosatellites.

REFERENCES

[1] Booch G.,'Object-Oriented Analysis and Design with Applications' ,Benjamin/Cummings,1994.

[2] PanosFitsilis,Vassilis C. Gerogiannis,Leonidas G. Anthopoulos,'Role of unified modelling language in softwaredevelopment in Greece – results from anexploratory study',IET SOFTWARE, 2014.

[3] Mark Read,Paul S Andrews,Jon Timmis,VipinKumar,'Modelling biological behaviours with the unified modelling language: an immunological case study and critique',JOURNAL OF THE ROYAL SOCIETY INTERFACE,2014.

[4] Peneva J., Ivanov S. and TuparovG.: 'Utilization of UML in Bulgarian SME – Possible Training Strategies', International Conference on Computer Systems and Technologies –CompSysTech, 2006.

[5] Gu, V.C., Cao, Q. and Duan, W.: 'Unified Modelling Language (UML) IT adoption — A holistic model of organizational capabilities perspective', Decision Support Systems, 2012, 54, (1), pp.257-269.

[6] Fuentes, L. and Vallecillo, A.: 'An Introduction to UML Profiles', The European journal for theInformatics Professional, 2004, 5, (2), pp. 6-13.

[7] DobingB. and Parsons J.: 'Dimensions of UML diagram use: A survey of practitioners',Journal of Database Management, 2008, 19, (1), pp. 1-18.

[8] Dobing, B. and Parsons, J.: 'How UML is used', Communication of the ACM, 2006, 49, (5), pp.109-113.

[9] Batra, D.: 'Unified Modelling Language (UML) Topics: Cognitive Issues in UML Research',Journal of Database Management, 2009, 20, (1), pp. i-x.

[10] Batra D.: 'Unified Modelling Language (UML) topics: The past, the problems, and the prospects', Journal of Database Management, 2008, 19, (1), pp. i-vii.

[11] Agarwal, R. and Sinha, A.P.: 'Object-oriented modelling with UML: a study of developers' perceptions', Communications of the ACM, 2003, 46, (9), pp. 248–256.