

Generic Process Model Structures: Towards a Standard Notation for Abstract Representations

Alta van der Merwe
School of Computing
University of South Africa
P O Box 392, UNISA, 0003
+27 12 4296339
vdmeraj@unisa.ac.za

Paula Kotzé
School of Computing
University of South Africa
P O Box 392, UNISA, 0003
+27 12 4296677
kotzep@unisa.ac.za

Aurona Gerber
Meraka Institute
CSIR
P O Box 395, Pretoria, 0001
+27 12 8413595
agerber@csir.co.za

ABSTRACT

The identification of process model structures is usually complex and costly. If these structures can be reused across boundaries, this could not only benefit the internal structure of one application domain, but could also benefit organizations where it is not feasible to initiate expensive process re-engineering innovations. Furthermore, a reusable process is not worth much if the process is not available. The preservation and availability of objects are therefore important, through libraries in the case of objects, or repositories in the case of process models. The creation of the MIT Process Handbook was a step in this direction. However, although the authors used object-oriented concepts in the abstract representations, they did not rigorously apply object-oriented concepts in the abstract representations used in publications on their process repository. Especially in the notation used and reference to specializations, there are some inconsistencies. To address these issues, we suggest the use of polymorphism, where specializations inherit from the generic base process model, and the use of more formal object-oriented notation for defining specialization.

Categories and Subject Descriptors

D.2.9 Management *Software process model*; K.6.3 Software Management *Software process*.

General Terms

Performance, Design, Standardization, Theory.

Keywords

Reusable process models, process model repositories.

1. INTRODUCTION

According to the IEEE glossary, reusability refers to the “degree to which a software module or other work product can be used in more than one computing program or software system” [6]. The preservation of objects for reuse is nothing new; the earliest form of reuse of information is the stories told and re-told for generations. Books were the next form of storing information for reuse and until very recently, the only way to preserve

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAICSIT 2007, 2 - 3 October 2007, Fish River Sun, Sunshine Coast, South Africa
Copyright 2007 ACM 978-1-59593-775-9/07/0010...\$5.00

information. With the computer revolution starting in the 1950s, a new form of preservation evolved through data storage on computer disks. The most popular way of storing and accessing data is still through the use of databases, e.g. student records in a university or patient records at a hospital.

In programming languages, the sensible reuse of program code is an innovative way of reducing costs, which not only reduces the cost of development, but also increases reliability and the effective use of specialists, and enforces standards [13]. In a programming environment, reuse refers to ‘the use of some pre-existing product, e.g. existing requirements, design, code, test software, and documentation’ [4:395]. A function or piece of code developed for one application is stored and made available for reuse by programmers as part of other program developments.

In the business application domain, researchers at the Massachusetts Institute of Technology (MIT) grasped the value of reusability and introduced the notion of reusable process model structures through the building of business process repositories [8]. The abstract representation of the process repository was developed during the early 1990s in the form of a Compass Explorer and in the mid 1990s the Phios software used for data access and manipulation of the process model structures was released [12]. The concepts used is described in detail in a Process Handbook by Malone, Crowston and Heman [9] as a model that:

- Uses object-oriented concepts for the preservation of the process model structures.
- Supports the notion of specialization and generalization.
- Supports the identification of generic process model structures for reuse by more than one company.
- Provide tools to access the process repository using the web.

The representation that Malone *et al.* [9] uses to construct the MIT Process Handbook, is based on the notion of specialization of processes from an object-oriented programming perspective and on the management of dependencies from a coordination theory [8] perspective.

The problem with the approach followed is firstly that in publications that describe the approach [8, 9] the authors use object-oriented concepts but do not represent models in object-oriented notation. One of the many advantages of consistent use of a modelling notation standard is the consistent interpretation thereof. Modellers would attach the same meaning to a model, and derive the same conclusions. This leads to reusability of models and representations. We recognise that Malone *et al.* [9] are traditionally not from a software development paradigm environment. However, when applying concepts from an agreed

standard, in this case the Unified Modeling Language (UML) and object-orientation, care must be taken to adhere to the original intention thereof, even if it is applied in a different domain. UML, published by the Object Management Group (OMG) is widely accepted as a standard notation for object-orientated notation [10, 11]. The same argument would hold for a researcher in information systems using an accepted mathematical notation. It is not acceptable to argue that one could borrow concepts from a model in one domain, but apply it using a different notation just because the application domain is different.

Furthermore, the way that inheritance is used in Malone *et al.*'s [9] approach, is in contrast with accepted rules of inheritance in object-orientation. We are aware that object-oriented languages often implement generalization/specialization differently. For the purpose of this paper, we propose to adhere to UML as the standard language used to model object-oriented concepts according to the OO-paradigm, as specified by the OMG.

In section 2, we give a brief overview on the components used in the process repository. In section 3, we follow with a discussion on the problems encountered with the current notation and use of inheritance, followed by some suggestions in section 3. In section 4 we give a brief overview on comments on the adapted changes to the process model structure by field specialists, followed by some comments from using the adapted abstract representations in a case study at the University of South Africa (UNISA) in section 5. Section 6 highlights some advantages of using the adapted repository abstraction and section 7 provides some concluding remarks.

2. THE PROCESS REPOSITORY

A repository is described as a place where data is stored. It could be in a database or as files, and could be distributed over a network or directly accessible to the user without using a network infrastructure. There are three important concepts in the building of a repository: the *abstract representation* used, the *physical storage* of the data, and the *software* used to access and view the data.

For *abstract representations* we draw schemas or models to present the structure of the data. For example, in a database environment an entity relationship diagram is used to show the entities and the relationships between the entities [3]. In the process repository the authors refer to a representation when they discuss the structure used in concepts. In this paper, we adapt the use of the word 'representation' when describing the elements and relationships between the elements in abstract representations. The *physical storage* of data is done on data storage devices such as hard drives and typically managed through *software*.

For the process repository [9], these concepts are illustrated in Figure 1, where the process model representation is used as a guideline in the development of the physical structures, which are in turn accessible through the Phios software from a computer system.

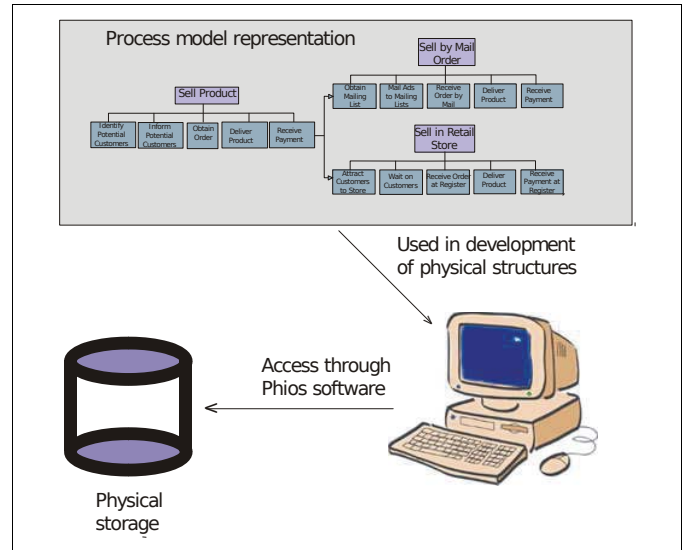


Figure 1. Components in the process repository

In this paper we focus mainly on the *abstract representations*, where the Process Handbook [9] suggests the use of a 'process compass' with two dimensions for analyzing business processes. The vertical dimension distinguishes between the *parts* of the process, while the horizontal dimension distinguishes between the different *types* of a process (Figure 2).

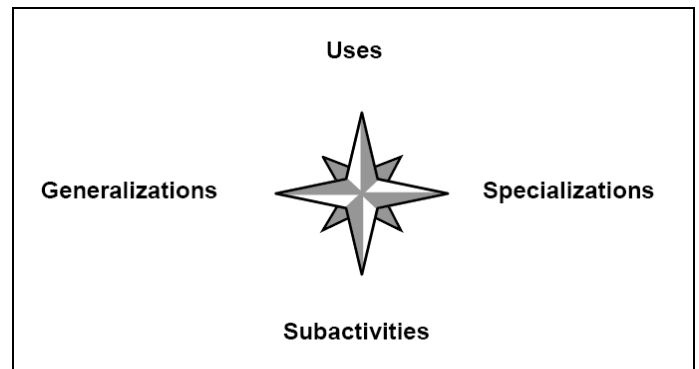


Figure 2. Compass Explorer [12]

2.1 Concepts in the Process Repository Representation

The process repository representation uses the specialization concept to show how process models can be inherited. The process repository representation extends existing process mapping techniques and, not only uses the break-down of a process into subprocesses or *parts*, but also defines different *types* for the process. Authors involved in research in the process repository regularly use the *Sell Product* example to describe the process repository representation for specialization of the processes [7-9, 12]. The process representation of *Sell Product* is given in Figure 3.

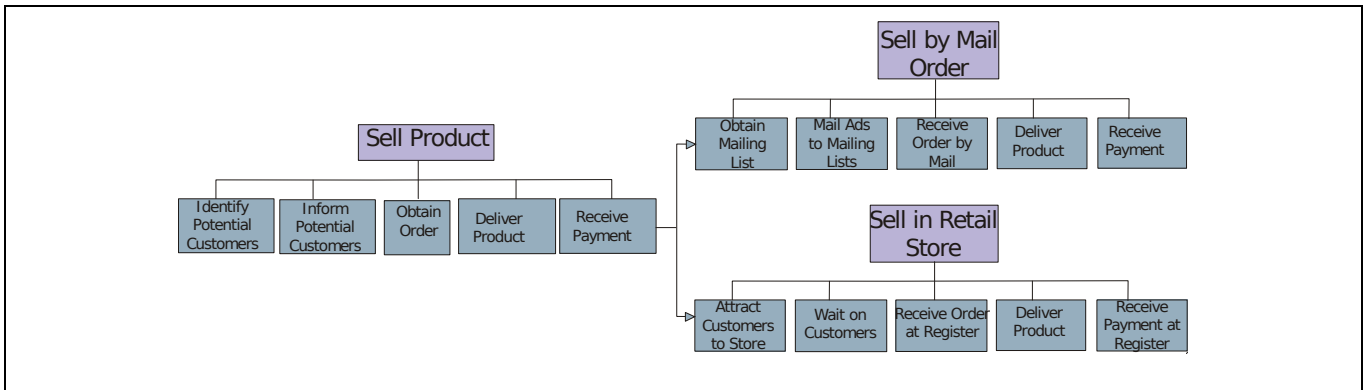


Figure 3. Generic sell product [9]

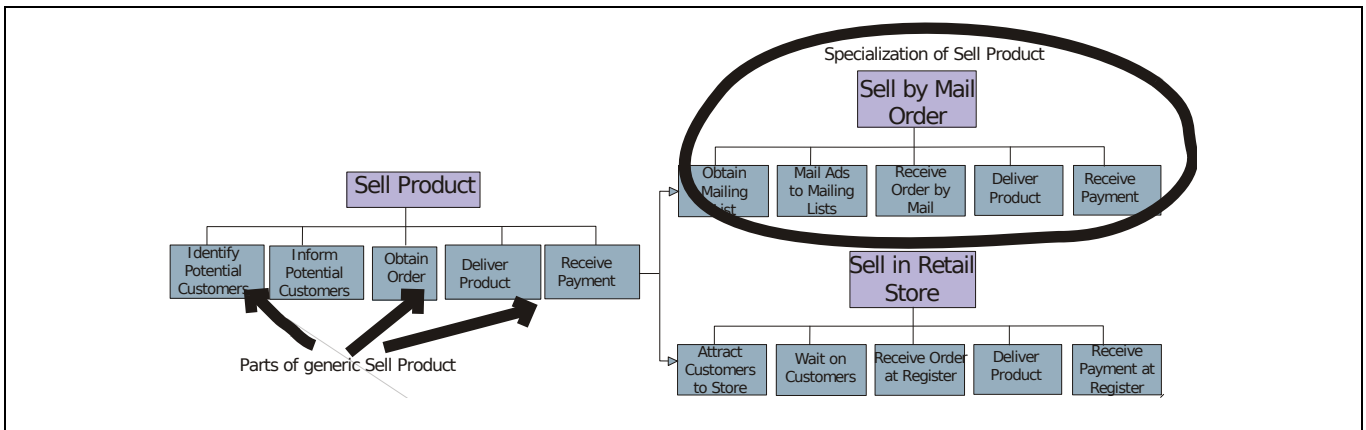


Figure 4. Parts and specializations

In this representation the *Sell Product* is broken down into *parts*, also called ‘subactivities’ or ‘subprocesses’. The subprocesses include the identification of potential customers, to inform potential customers, to obtain an order, deliver a product and to receive payment. For each generic process representation (such as *Sell Product*) it is also possible to map the representation to special cases of the process. For example, *Sell by Mail Order* and *Sell in Retail Store* are examples of special cases for the generic *Sell Product* (Figure 4).

The concept that the process repository supports is based on *inheritance* used in object-oriented development. According to Firesmith and Eykholt [4:203] inheritance is the ‘incremental construction of a new definition in terms of existing definitions without disturbing the original definitions and their clients’. In inheritance, the child class (subclass) inherits the properties from the parent class (superclass). For example, in an IT company employees could either be full-time employees or contractors. In the case of full-time employees the employee will receive a salary. In the case of a contractor, the employee will receive a payment at the end of the month based on his hourly wage and the hours that he worked (Figure 5).

In this example, the subclasses *Full-time employee* and *Contractor* inherit the Number, Name and Contact details from the superclass *Programmer*. The *Full-time employee* also has an additional attribute Salary and the *Contractor* includes two additional attributes, Hourly rate and Hours worked. The *Full-time employee* and *Contractor* are called specializations of

Programmer. If the diagram is read from the top-down, object-orientation refers to the concept of generalization. Generalization is the ‘process of creating a generalization from one or more specializations’ [4:183]. In our example, the *Programmer* is a more general element than the *Full-time employee* or the *Contractor*. Therefore, the *Programmer* is a generalization for *Full-time employee* and *Contractor*.

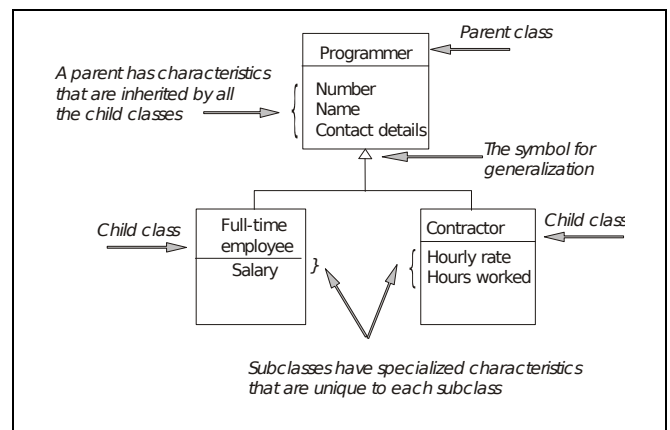


Figure 5. Employee types in an IT company

Therefore, in the process repository representation the *Sell by mail order* and the *Sell in retail store* inherits the *Sell Product* from the parent. Both are specializations of *Sell Product* and it

is possible to deduce that *Sell Product* is the more general structure, or the generalization. There are, however, two problems with the way that the structure is presented.

2.2 Problem 1: Notation Used in Process Repository

The first problem with this model is that authors used object-oriented concepts but do not represent the model in object-oriented notation, in this case, UML.

In the 2001 UML specification [10:3-86], generalization is 'shown as a solid-line path from the child (the more specific element, such as a subclass) to the parent (the more general element, such as a superclass), with a large hollow triangle at the end of the path where it meets the more general element' (Figure 6).

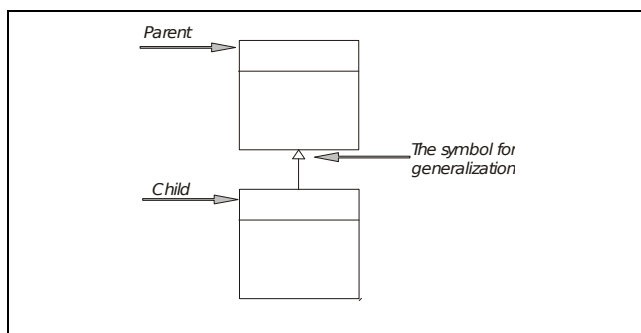


Figure 6. Generalization relationship

Note that the hollow triangle points towards the more general class, or the parent. In the notation used by Malone *et al.* [9], and also graphically illustrated in Figure 3 and Figure 4 the arrow points away from the parent. The danger of not using the UML notation standard consistently is that it may lead to misunderstanding of the abstract representation.

2.3 Problem 2: Changes in the Specialization

Another difference between true object-oriented use of inheritance and the process repository representation is that the process repository representation allows changes to the parts of the specialization. To describe this in more detail, it is first necessary to look at the notation used for a class in an object-oriented environment (Figure 7).

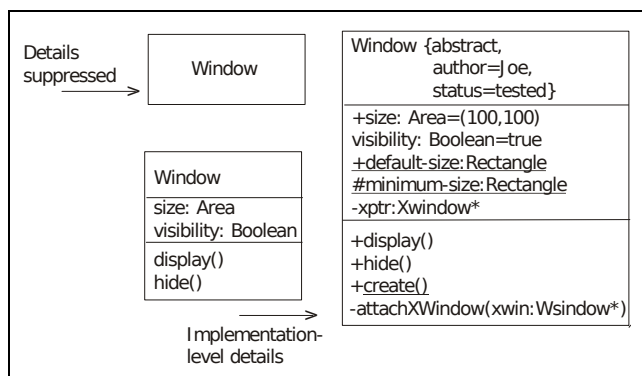


Figure 7. Class Notation [10:3-37]

Process models relate to the analysis level of the class notation where the data and methods are displayed in the class. In the

example above, the *Window* class has two attributes, size and visibility. It also has two methods, display() and hide(). If a subclass inherits from this class, it will inherit all the attributes and the methods. For example, if there are two subclasses *Blinking_window* and *Wave_window* for the *Window* class that display a window on the screen, both these will inherit the ability to display and to hide (Figure 8).

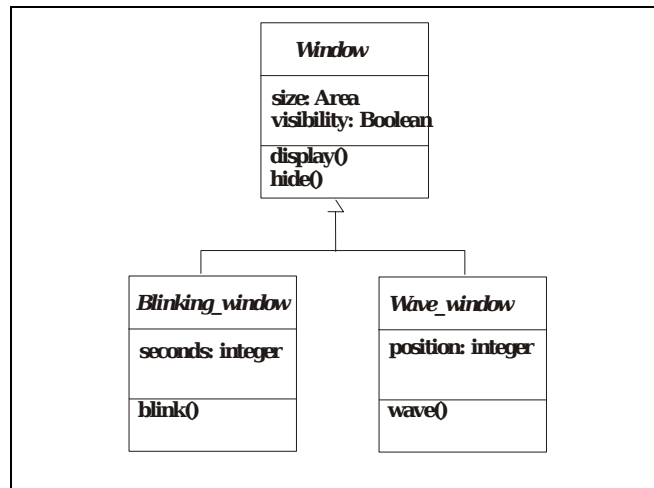


Figure 8. Two subclasses inherit methods from *Window* class

In the example, the *Blinking_window* subclass will also be able to 'blink' and the *Wave_window* will be able to 'wave'. The programmer is allowed to add methods and attributes to the subclasses and he is allowed to change the way that the two windows are displayed and hidden (methods inherited from the superclass), but he is not allowed to change the function of the method. If the function was to display the window, the window must still be displayed, irrespective of the inner workings of the program manipulating it to display. The result should only be a window that is displayed on the screen.

In the process repository example, the authors allow a change to the function of an inherited subprocess. For example, *Sell in Retail Store* inherits from *Sell Product* the subprocess *Wait on customers*. The function in the original process structure was to inform clients, which is done in the *Sell by Mail Order* specialization. But in the *Sell in Retail Store* specialization the function is not to inform, but to wait. This is a change in the original intention of the subprocess (Figure 9).

3. SUGGESTED IMPROVEMENTS

In a specialization hierarchy, the objects (or processes) inherit the features of their parent and modify them incrementally, promoting comprehensibility, maintainability and reusability [15]. Furthermore, the use of a process hierarchy also supports the generation of design alternatives and suggests an organizational framework where relevant processes could be sought [8]. The model in the MIT Process Handbook is based on specialization and generalization taken from the object-oriented paradigm. We suggest two modifications to the process repository, the use of polymorphism, where specializations inherit from the generic base process model, and the use of more formal object-oriented notation for defining specialization.

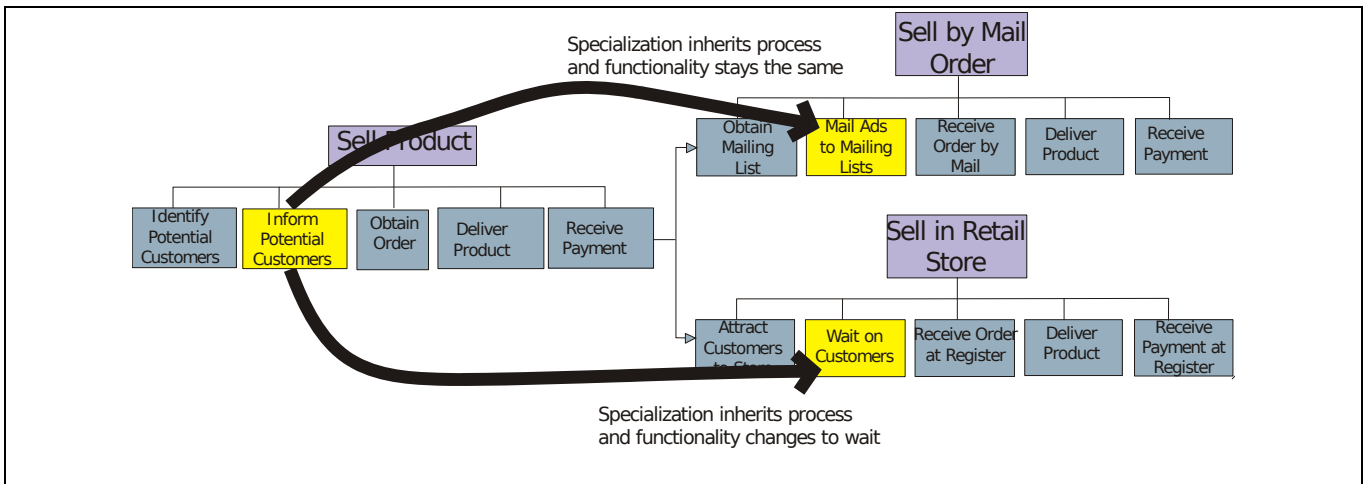


Figure 9. Specialization changing the function of the inherited process

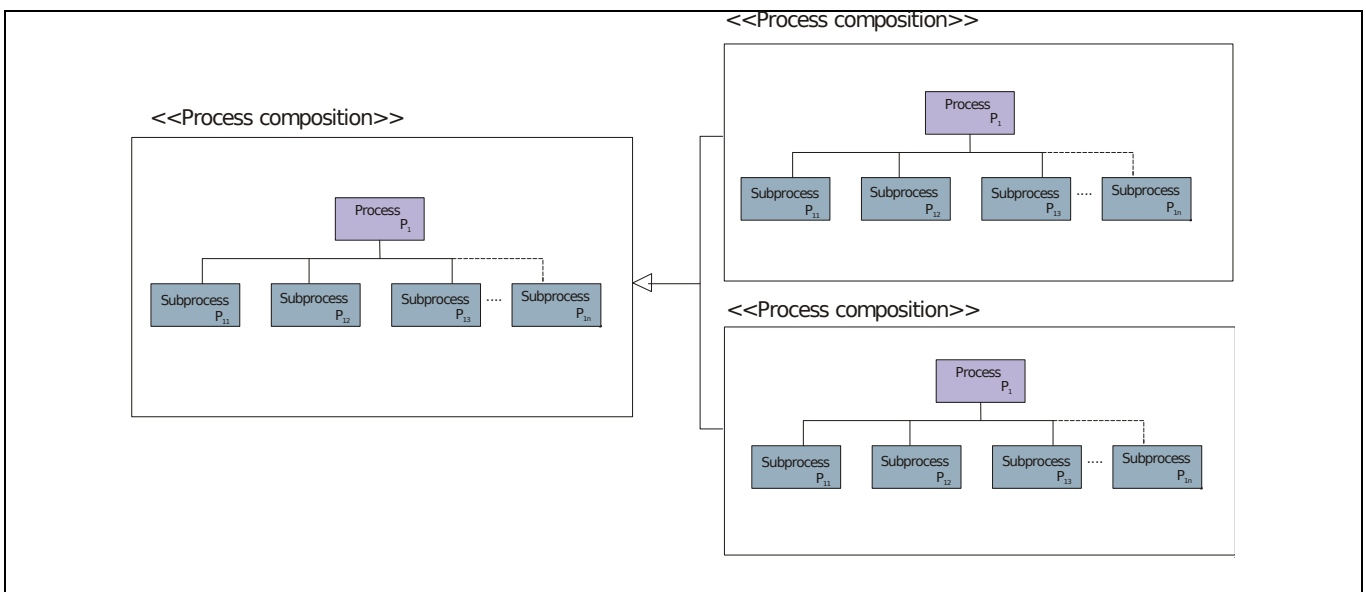


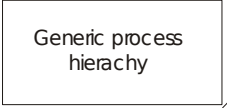
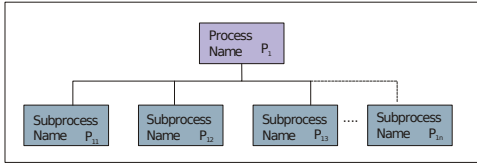
Figure 10. Suggested notation for specialization in the educational process repository

3.1 Suggested Improvements with regard to Notation

As stated above, the MIT Process Handbook approach claims to use generalization and specialization from the object-oriented paradigm. Our analysis, however, revealed that according to the *Sell Product* example used in most of the published papers on the process repository, the notation does not agree with the notation used for specialization in the object-oriented paradigm. In object technology, the arrow shows from the child object to the parent and not as in this example, where the arrow shows from the generic process to the specialization. We believe that the notation in this model may not have a significant meaning as the notation is never discussed in the papers where the authors refer to this example [1, 12].

We therefore suggest an adaptation of this model to support the notation used for generalization and specialization in the object-oriented paradigm, with the arrow pointing to the generalization and not the other way around. In Figure 3, 4 and 9 the arrow shows from the parent to the specializations and in the adapted model, Figure 10, it goes from the specializations to the parent.

Furthermore, we also suggest the use of a new stereotype called the *Process Composition Stereotype* to formalize the specialization between the generic process and the representations. Stereotypes are used to extend the existing object notation and therefore formalize the model within the object-oriented paradigm. The description for the Process composition stereotype is the following:

UML Metaclass Extended	Class
Semantics	The generic process consists of one or more subprocesses used to derive the goal for the process. If only one subprocess, the process is called 'atomic'.
Constraint	Must produce at least one output.
Diagram Notation	The notation used is <<process composition>>.
Predefined process composition	<p>In a diagram the process composition is described by a rectangle with the generic process and subprocesses drawn in the rectangle as a process hierarchy.</p> <pre><<Process composition>></pre>  <p>Each generic process composition consists of a generic process with a subset of subprocesses. In a specialization, polymorphism is applied – the method of reaching the goal of the subprocess may differ, but the output stays the same. In a specialization, subprocesses may be added.</p> <pre><<Process composition>></pre> 

3.2 Suggested Improvements with regard to Inheritance

Consider the generic process P_1 with subprocesses P_{11} to P_{1n} (Figure 11).

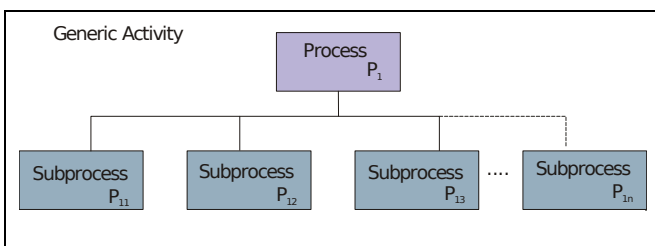


Figure 11. MIT Process Handbook process model

The MIT Process Handbook specifies that: 'Each activity *inherits* automatically the subactivities and other properties of its generalization, except where the specialized activity *adds* or *changes* a property' [12:15]. The implication in the model above is that the model may be extended (properties may be added to include another subprocess $P_{1(n+1)}$) or any property of the subprocesses may be changed (P_{12} may be changed to another process P_{kl}).

As mentioned previously in section 2.2, this is in contrast with the rules in object-orientation. The implication is that the process repository representation does not support the concept of *polymorphism*, which specifies that the output of an inheritance should stay the same (even if the methods change).

In the object model, using the concept of polymorphism allows the user to change the way in which a method arrives at the desired output, but the output stays the same.

We suggest that the rule that applies to polymorphism to the effect that the output of a subprocess should stay the same, should be enforced. As a result, fewer subprocesses will be included on a higher level, where all the subprocesses have the same goal as the subprocesses in the base model. In other words, the specialization inherits the original subprocesses from the generic model and, if necessary, subprocesses can be added to the specialization. The suggested change to the structure is illustrated in Figure 12.

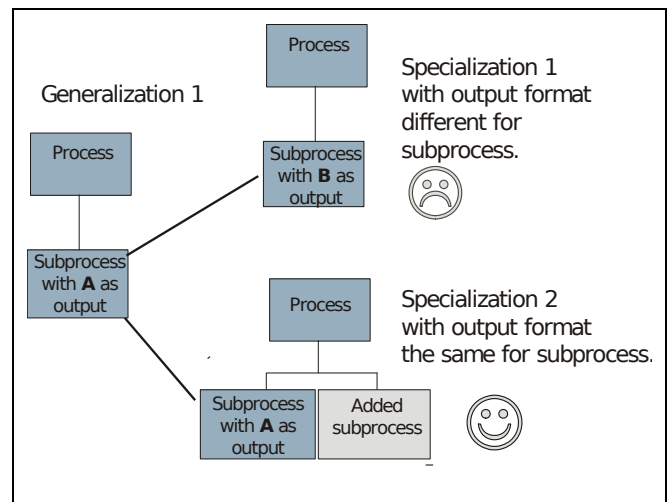


Figure 12. Inheritance and additions of subprocesses

The adapted model suggests that Specialization 1 is not allowed, where the output of the subprocess differs from the output of the parent, but Specialization 2 is allowed where the output of the subprocess has the same form as in the parent abstraction.

4. FEEDBACK FROM FIELD SPECIALISTS

As a triangulation exercise we wanted to know how the adapted model would be perceived by peers. The model was discussed with 3 practitioners using object notation in development projects at different software development houses and 3 staff members at UNISA that is actively involved in courses using object-orientation concepts. The three practitioners were all previously involved in the use of process models in development of systems. The staff members at UNISA were all involved in courses related to Structured System Analysis and Design and Object-Oriented System Analysis and Design.

The adapted model with the implications (Figure 13) was used in information interviews to discuss the notation and the limitation on specializations.

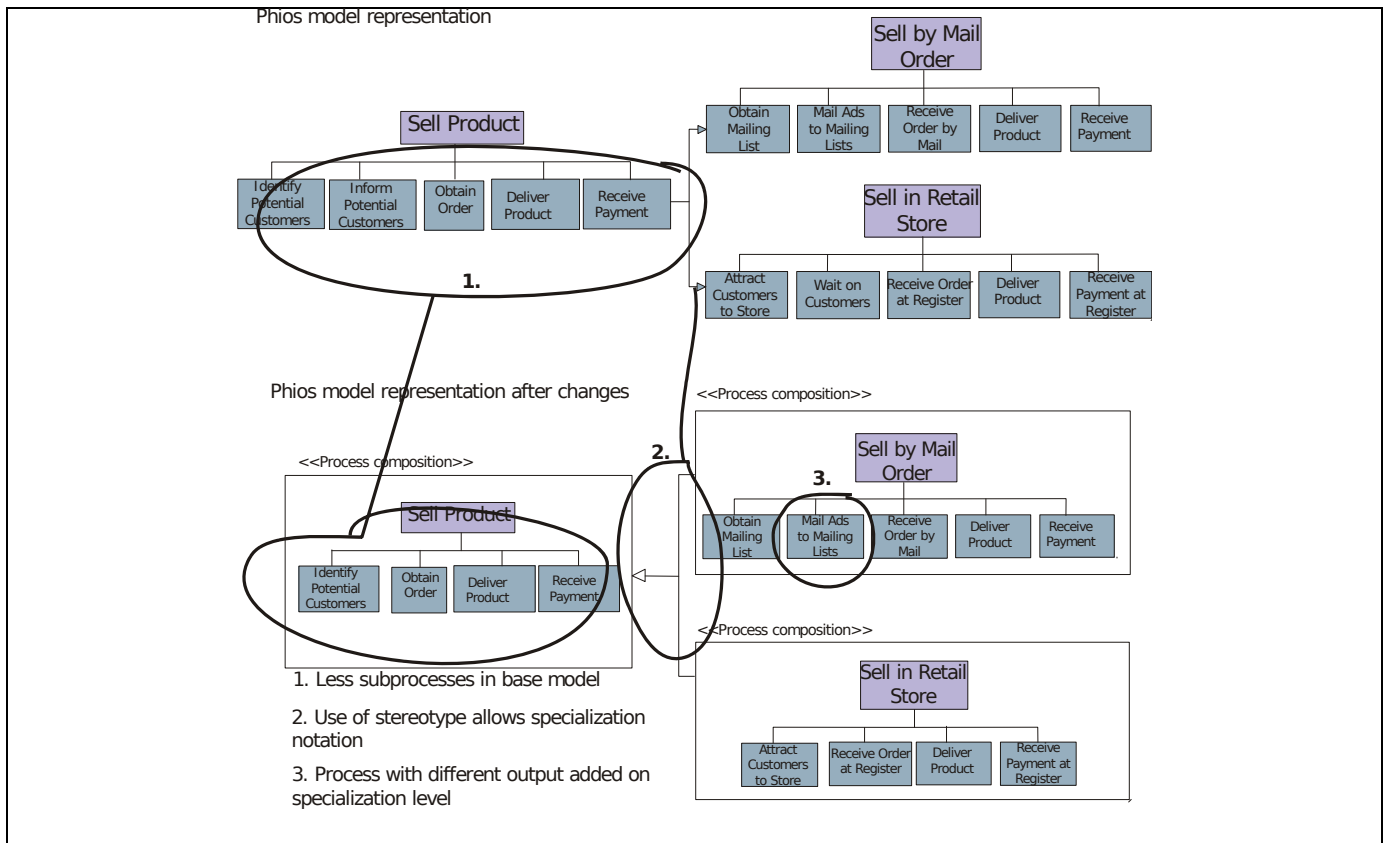


Figure 13. Sell Product in the process repository and using the suggested adaptations

The following are significant comments recorded during discussions with interviewees:

- The use of process models in the object paradigm is an unfamiliar concept.
- If one wants to use concepts from the object paradigm such as specialisation and generalization it is necessary to select a notation (preferably a standard notation such as UML) and define the way in which concepts will be used.
- The stereotyped notation defined and the enforcement of the polymorphism rule with regard to the output of subprocesses in a specialization are only moves in the direction of a more formal notation of the environment. More research is needed on sequence of execution and information lost in the diagrams such as the input and outputs associated with each process.
- Some comments were made on the nature of the implementation of polymorphism in applications. One interviewee claimed that it is possible to change the output of a specialization when used in combination with dynamic bounding. The problem is that this is not true to the object paradigm. He did agree that this is an advanced topic and should be handled as an exception rather than a rule. Therefore, this should not be enough reason to be lenient

when using object notation such as specialization and generalization in this application domain, and it therefore does not apply to the abstract level of process models as suggested by the adapted model.

The concerns raised by the respondents were the same as our own which was the motivation for suggesting the adaptations to the Phios process model representation. This confirms the proposition that the suggested notation is a better representation in an object environment from both a theoretical and practical perspective.

5. A CASE STUDY: UNIVERSITY REGISTRATION SYSTEM

The suggested adapted representations were used in a specification document during a reengineering effort at the UNISA, which formed part of a PhD study [14]. In this reengineering effort, the preservation of educational process model structures were investigated, and the process model structures were composed using the new adapted model introduced in section 3. An example of one of the process compositions is the UNISA application process, illustrated in Figure 14.

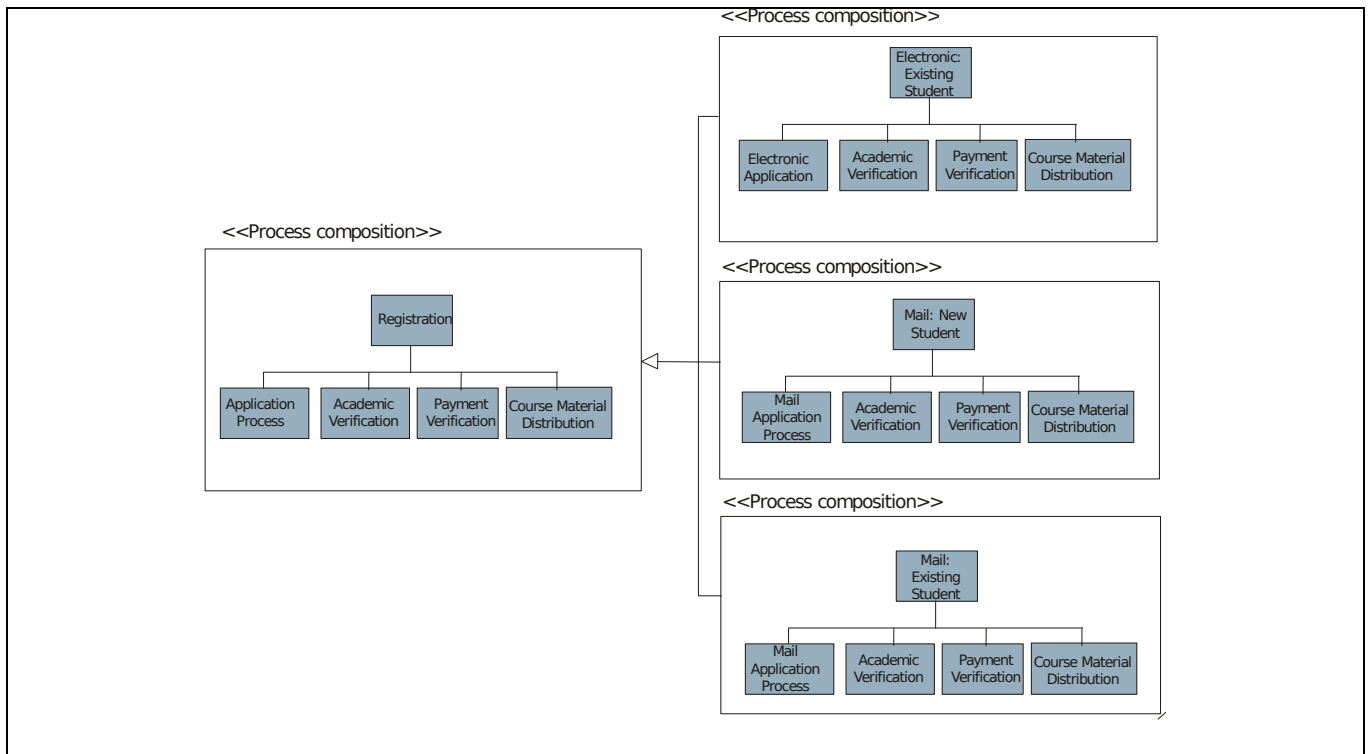


Figure 14. Specializations for UNISA Application Process

One of the biggest advantages is the *extensibility* of the model. Any user of the model may extend it to include new subprocesses according to new specializations. For example, the focus of the case study was based on undergraduate studies; if the user wants to add a registration that is for postgraduate students only, it could be implemented easily by adding a new specialization for the generic REGISTRATION process. This specialization will then inherit the four generic subprocesses defined for the REGISTRATION process. The developer only needs to map these processes according to his pre-knowledge on the application domain and decide whether the processes are sufficient or whether an additional process is needed. This emphasizes another characteristic of the repository model, namely its *reusability*. The specialization of a generic process model enables the developer to *reuse* what has already been identified previously and extend only if needed.

The *maintenance* of the process model repository is uncomplicated. Processes can be added at any time to describe a specific specialization. A problem that should be addressed is the sequence of execution of processes on the same level. It is assumed that the sequence of processes is from left to right in the representation of the educational process model. If a set of processes is inherited for a new specialization, there may be a process that is added between two existing processes. If the developer is not aware of the sequence of process execution, a model that is not a real representation of the real world could easily be created.

The use of an accepted *object-oriented notation* for presentation of the specializations enhances the usability of the models. If a notation is used that is accepted generally as a standard notation by different role players in development, the 'language' for

discussions is the same and the developers can focus on the solutions and not on what the current environment actually looks like. In implementing the adaptations of a more formal way of representing the specializations through stereotypes and the use of polymorphism, this model moves in the direction of supporting a standard notation. The use of accepted standard notation implies that this model supports more characteristics of the object notation than the previous model does.

In the object-oriented paradigm, the models used should be easy to understand, easily maintained, support object-oriented modelling concepts, be information-rich to model different concepts and be reusable [2, 5, 10].

6. ADVANTAGES OF USING THE ADAPTED REPOSITORY REPRESENTATIONS

From the feedback received from object specialists and using the adapted representations in a reengineering effort, it is possible to summarize that the adapted model conforms to the following"

- *The model is understandable:* The goal of models is to make the 'picture' clearer for the reader using it as a reference tool.
- *The model is easily maintainable:* Using the generic process with specializations allows the user to add processes on lower levels if the higher level neglects a needed process.
- *The model supports object-oriented modelling concepts:* We suggested the use of polymorphism and stereotypes to make the model more object-oriented. The creators of the process repository suggested the use of generalization and specialization from the object-oriented paradigm without

using the object-oriented notation in their own models. From the interviews it was confirmed that the adapted model supports more object notation than the initial model does.

- *The model is information-rich:* The adapted model gives information on the parts and the specialization of the environment. Using the model will enable the reader to derive logical arguments on the generic process models and on the parts represented in different scenarios. This was not added to the model but only confirmed as being an advantage of the model in general.
- *The model is reusable:* The generic process model and the specializations thereof can be used and reused because of the generic characteristic of the models. Simply by discussing the models with the respondents, the models were already used as a reference model.

7. CONCLUSION

The creation of the MIT Process Handbook was a giant leap towards the creation of an environment where process model structures can be preserved in a process repository environment. However, although they used object-oriented concepts in the abstract representations, they did not rigorously apply object-oriented concepts in the representations used in publications on their process repository.

To address these issues, we suggest the use of polymorphism where specializations inherit from the generic base process model and the use of more formal object-oriented notation for defining specializations.

As a case study, the abstract representations for a university registration system were created using the suggested changes. The adapted models were easy to understand, easily maintained, support object-oriented modelling concepts, information-rich to model different concepts and reusable

8. REFERENCES

- [1] Bemstein, A., Klein, M., Malone, T.W., *The Process Recombinator: A Tool for Generating New Business Process Ideas*. In: T.W. Malone, K. Crowston, and G.A. Heman, (eds.): *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, London, England, 2003.
- [2] Booch, G., Jacobson, I., Rumbaugh, J., Rumbaugh, J.: *The Unified Modeling Language User Guide*. Addison-Wesley, Boston, (1998).
- [3] Cardenas, A.F.: *Data base Management Systems*. Allyn and Bacon, Massachusetts, USA, (1985).
- [4] Firesmith, D.G., Eykholt, E.M.: *Dictionary of Object Technology; The Definite Desk Reference*. Sigs Books, New York, (1995).
- [5] Hamon, P.: *Objects In Action: Commercial Applications of Object-Oriented Technologies*. Addison-Wesley, Reading, MA, (1993).
- [6] IEEE: *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers, New York, NY, (1990).
- [7] Klein, H.K., Myers, M.D.: A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly, Special Issue on Intensive Research*. 23(1) (1999), 67-93.
- [8] Malone, T.W., Croston, K., Lee, J., Pentland, B., Dellarocas, C., Wynor, G., Quimby, J.: Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science*. 44(3) (1999), 425-443.
- [9] Malone, T.W., Crowston, K., Heman, G.A.: *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, Cambridge, Massachusetts, (2003).
- [10] OMG: *OMG Unified Modeling Language Specification*. (2001) [cited 2002 15 June 2002]; Available from: <http://www.omg.org/>.
- [11] OMG: *OMG Unified Modeling Language Specification version 2.0*. (2003) [cited 2005 April 2005]; Available from: <http://www.omg.org/>.
- [12] Phios: *New Tools for Managing Business Processes*. [White Paper] (1999) [cited 1999 March 1999]; Available from: <http://www.phios.com/phioswhitepaper.pdf>.
- [13] Sommerville, I.: *Software Engineering*. 6 ed. Addison-Wesley, Boston, MA, (2000).
- [14] Van der Merwe, A.: *Towards a Reusable Process Model Structure for Higher Education Institutions (PhD thesis)*. Thesis, School of Computing, University of South Africa, Pretoria, 2005.
- [15] Wyner, G.M., Lee, J., *Defining Specialization in Process Models*. In: T.W. Malone, K. Crowston, and G.A. Heman, (eds.): *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, London, England, 2003.