APPLYING OBJECT-ORIENTED MODELLING TO SUPPORT PROCESS TECHNOLOGY

Fereidoon Shams–Aliee
Department of Computer Science
University of Manchester
Manchester M13 9PL U.K.

Brian C. Warboys
Department of Computer Science
University of Manchester
Manchester M13 9PL U.K.

ABSTRACT

Recent trends focusing on process technology applied in different domains confirm the obvious fact that a quality product can only be the result of a quality process. One of the basic requirements for improving the quality of a process is to manage its complexity by means of modelling techniques using different levels of abstraction. A simple yet powerful view for modelling a process is to represent the behaviour of its identified agents in terms of their activities and interactions with each other. To implement such a view, object-oriented modelling can be exploited for modelling the agents of a process in terms of objects and representing the dynamic behaviour of objects through the message passing and concurrent execution of these objects within a process.

In this paper, the idea of employing object-orientation as a modelling technique for representing the behaviour of a process in terms of cooperative objects will be specified. The need to identify an abstract level, as an intermediate level between the object and process concepts, has been recognised and a new method for capturing such an abstract level has been developed. Based on this method the final structure of a process will be represented in terms of Petri nets. This will also represent the structure of the reusable components of a process in the form of patterns of cooperative objects, and the interface to be reused in different processes. The final model can be simulated through Petri net tools to check the specification of a process against its behaviour in terms of the cooperation of different objects playing roles within that process.

INTRODUCTION

A process view of a system is a view of the system's dynamic behaviour over time considering the system as a whole rather than focusing on individual activities. Current methods do not sufficiently capture abstractions that remain stable over time, and thus, do not establish a good baseline for long term process improvements. It has been advocated that an efficient process modelling approach has to focus not only on the analysis of data flow and transformations, but also on the communication and coordination of process elements (Curtis et al., 1992). Object-oriented design is believed to be powerful enough to satisfy such requirements.

Object-oriented modelling has been recognised as a powerful method for modelling in many software domains (Booch, 1994; Eliens, 1995; Rumbaugh et al., 1991). The idea that *software processes are software too* (Osterweil, 1987) implies the extension of the use of object-oriented modelling methods to process modelling. Recently object-oriented concepts have been used for improving process models in several projects from a number of different perspectives (Baldi et al., 1993; Belkhatir and Melo, 1994; Conradi and et al, 1994; Garg et al., 1994). The main focus in most of them has been the development of the notion of an object-oriented data-base to support the process model. Although some of these approaches have tried to use objects to represent structural views of a process, less attention has been given to applying object-oriented characteristics to represent the behavioural dependencies of a process from the point of view of applying object-oriented concepts as a modelling technique.

A simple yet powerful way to model a process is to represent the behaviour of its identified agents in terms of their activities and interactions. To implement such a view, object-oriented modelling can be exploited for modelling the agents of a process in terms of objects and representing the dynamic behaviour of objects through the message passing and concurrent execution of these objects within a process. However, more consideration needs to be given to the specification of processes based on objects with emphasis on exploiting objects to represent the communication and the coordination aspects as the major features of a process (Curtis et al., 1992).

In building process models based on objects, we have identified a gap between object capabilities and the process world. To overcome this problem, a level of abstraction, named *role*, above the object level, has been defined. We have represented roles in different levels of abstraction containing CRC (Class, Responsibilities, and Collaborators) cards, formalised CRC (CRC plus path expressions), Petri nets, and encapsulated structures to provide a pattern representation of roles. Therefore, the focus of this work is to define an intermediate level between the process and object levels. In order to represent the usefulness of defining such a level of abstraction, an approach to capture, refine, and extract an enactable yet formal model of processes based on objects has been developed.

THE IMPORTANCE OF OBJECTS IN DESIGNING PROCESS MODELS

The importance of objects as the basic structure for process models from a modelling point of view can be classified into different perspectives such as structural and behavioural views of system architecture, reusability and encapsulation as desirable features in design methods, communication and concurrency as important factors of any complex system, and finally modelling and formality as providing simple but accurate views of a process model. We now briefly discuss the importance of these different aspects when modelling a process based on the notion of the collaboration of objects.

- Structural & Behavioural

  There are two related aspects of system architecture: structure and behaviour. Since a structural view point of a system represents the decomposition of a system into its components and the relationships amongst the components, objects can be seen as clearly recognizable components of a process. Without the assumption that a system or a process is composed of different components, it would be hard to represent the behaviour of that system or process (Milner, 1989). Therefore, representing the system's components in terms of different objects provides the capability for viewing and designing the structural relationships between these objects. This can be identified as the basic requirement for the modification of a process through the changing of its structure.

  The behavioural view point is the temporal sequencing of the activities performed by a system (done within a process) and the communication patterns among its components that are necessary for the system to achieve its objectives. The activities of a component will depend on the context in which a component has been used. Objects can provide such different behaviours during their life cycle. By changing the role of components in different contexts, different aspects of a component at different times can be represented through the execution of an object (Belkhatir and Melo, 1994). This can provide the basic capability for analysing and studying the execution of a process over a specific time.

- Reusability & Encapsulation

  By viewing a process in terms of the collaboration of encapsulated components such as objects we have provided the basic modularity required for the reuse of a component in a number of different processes. This view will also improve the security of the components which finally result; the security improvement of the whole process being realised through the encapsulation of these different secure components. Representing the notion of the state of a process by means of the combination of the different states of different objects is another direct result of adopting the notion of objects within a process. By adopting this view we have extended the notion of reusability from the object level to the process level to some extent.

- Communication & Concurrency

  Communication and concurrency are complementary notions, both essential for understanding the complex nature of dynamic systems. Underlying both these notions is the assumption that each of the several parts of a system has its own identity, which persists through time (Milner, 1989). Therefore, such parts can be mapped into objects and through the interaction of different objects, in terms of message passing, we will be able to represent communication amongst these parts. Each action of an object will be either an interaction with another object, and thus a communication, or it can occur independently and thus cause concurrency with other objects. In this way, objects will help in specifying coordination and cooperative work which is an essential feature to be addressed for many process modelling applications. Managing (handling) the complexity through the understanding of the concurrent execution (multiple thread) of the components of a process is another advantage of employing objects to represent the concurrent aspects of processes.

- Modelling & Formality

  Many concepts in real processes such as humans, tools, and resources can be modelled by objects thus facilitating the mapping of real world concepts to the modelling concepts for the purpose of process modelling. Such a mapping can improve the understanding of what is going on in terms of the actions and reactions of different components of a process by viewing their behaviour within a process. This extends the readability of a process from the modeller point of view to the process viewer or the end-user point of view.

By identifying objects we can focus on the smaller components rather than being forced to focus on the whole process as is frequently the case for many modelling tools including Petri net-based approaches. Moreover, providing the possibility of viewing a process from different perspectives by changing or modifying the definition and task of different objects from time to time can be achieved. This is usually required for a process model and has been accepted as one of the advantages of the object-oriented view.

An important requirement of a modelling method is to manage the complexity of a system by breaking it down into simple sub-systems. Exploiting the notion of objects will help to provide such a facility as we can then view a process in terms of the sub-processes which correspond to objects. Defining the dynamic behaviour of a process in terms of the formal behaviour of objects can provide for the possibility of proving the correctness of the whole process by aggregating the proof of every object used in that process. A formal specification of a process model based on a formal specification of component objects would also reduce the complexity of the formal proving of such a process.

PROBLEMS IN APPLYING OBJECTS TO PROCESSES

Objects can provide the basic requirements, although not enough concepts, to represent all the aspects of a process. Objects, in most applications, are only responsible for stating what they can provide or which services they present to the outside world and how one can use these services. However, processes need also to be concerned with at least two other aspects regarding the question of *who* and *when*. In answering *who*, the agents (human or tool), which take the responsibility of playing an action, should be determined and in answering *when*, the pre-conditions for any action to take place need to be decided. Although in some applications objects may also be considered to cover these aspects, in these circumstances they usually are concerned with the dynamic behaviour of objects over time rather than the definition of objects.

Such a distinction leads to the viewing of objects and processes in different dimensions. In the process dimension, components are mainly goal-oriented, because they need to be organised to show the progression of different steps to reach a goal. Hence, the dominant view in a process is the view of the transformation from one step to another step. In this view we usually look at different steps as different functions which have been organised to fulfill some partial goals. This view forces a process component to be meaningful only in the context in which that component has been used.

However, the object dimension is rather data-oriented. It views its responsibility in terms of the data and the procedures required to manipulate that data in order to provide some specific services. In the object view we usually wish to model the behaviour of a physical object or some other intangible entities from the real world. Although such objects may collaborate with other objects to accomplish their own task, they still can be viewed as potentially reusable components in different contexts.

Therefore, there is a gap between a process model and the object paradigm making it difficult to think about a process model initially in terms of objects. If we wish the notion of objects to be meaningful in the process dimension, we need to bring these two dimensions closer together by interleaving another interface between them which has the ability of being meaningful to both dimensions. By defining a new level above the pure object level, we are a step further towards the concept of process and yet retain the notion of object in this new level. Moreover, throughout an object-oriented system, groups of related objects often cooperate to perform some task, and the interaction patterns within a collaboration group are often repeated throughout a system, and even in different systems, with different participating objects. Wirfs-Brock and Johnson (Wirfs-Brock and Johnson, 1990) observed that reuse in object-oriented systems typically occurs by reusing groups of objects interacting to accomplish tasks. It has also been advocated that collaboration groups represent a reusable architectural abstraction above the level of individual objects that are essential for understanding, designing, and reusing object-oriented systems.

As the result of this introduction it seems that the concepts of framework (Johnson, 1992) in object-oriented methods can be associated with the concept of role in process modelling applications. This means that if we can recognise, capture, and represent a process based on such frameworks (roles) which in turn represent a pattern of communicating objects, then this would fill the reported gap and improve the level of reusability when building a process model based on objects. Groups of such objects would still be reusable in different applications. Therefore, we can define another abstract level which has the characteristics of both objects and the properties of processes.

UNDERLYING CONCEPTS TO CAPTURE AND MODEL PROCESSES USING OBJECTS

Defining Role as an Intermediate Level

Role definition is basically concerned with role theory as used for organisational coordination. Here, roles are the building blocks of organisational structures (Singh and Rein, 1992). The role concept has been used slightly differently in different contexts although basically with the same root meaning. In process modelling a role is a list of responsibilities assigned to an agent to be followed in a specific order. Such an agent needs to interact with other roles, to assist part of a process.

From an object perspective, roles prescribe the way we need to use objects to approach the system's goal. As defined in(Booch, 1994), a role is the face an object presents to the world at a given moment or a particular aspect, or behaviour which an object exhibits during a period of time (Arapis, 1990). The most important point is the possible relationship between role and object concepts which provides an insight into viewing roles in terms of cooperation amongst different objects to play those roles. Figure 1 represents a many-to-many relationship between the role and the object concepts. A person as an object may play roles of driver, customer, parent and so on. Further in playing a role such as librarian, a person object, a book class object, a file object, and some other objects can be used.

In designing process models, roles are the basic building blocks of processes (Warboys, 1989). In contrast with other software applications, role can be seen as a small application reusable in different processes. Instances of roles in a software process are project manager, design engineer, and QA engineer. When viewing such roles we naturally have to view the objects involved in playing the role. For example, depending on the level of detail that we wish to model, some objects which cooperate to play a project manager's role are project manager as a person object, documents, files, E-mail tool, and so on. In (Wiernga and de Jonge, 1993) a precise distinction between role and object concepts has been made and some good examples are provided to identify the differences between these concepts.

The role concept represents an abstract level above the object level which shows the behaviour of a component of a process and is suitable for use in designing process models. Therefore, a role can be viewed as a framework for representing the component of a process based on its behaviour in terms of the cooperation of one or several objects in playing that role. This also prevents the generation of new problems when trying to use objects directly for modelling processes.

By applying objects to play a role, the notion of role can retain the advantages, such as flexibility for change and maintainability, that a good object-oriented design possesses. This influences the quality of process models as they are composed of robust, reusable, and rather flexible components. Thus, the notion of reusability has been extended from the level of object to the level of role and to some extent to the level of process. Further, Viewing roles before objects is the way to develop a high level design without committing to the existence of particular objects and the classes that support them.

Characterising Roles

Class, Responsibilities, Collaborators (CRC) cards were invented by Cunningham in response to a need to document collaborative design decisions (Beck and Cunningham, 1989). CRC method has been recognised as a useful method for building structures of communicating objects (roles) in many object-oriented applications (Andersen and Reenskaug, 1992; Buschmann, 1993; Cain and Coplien, 1993; Reenskauge and et al, 1992; Wirfs-Brock et al., 1990). CRC definitions are very close to capturing the nature of a role since they are defined using a responsibility-driven approach rather than a data-driven approach (Wirfs-Brock and Wilkerson, 1989).

Figure 2 represents a CRC index card. The class name of an object creates a vocabulary for discussing a design. Responsibilities identify problems to be solved. The solutions will exist in many versions and refinements. A responsibility serves as a handle for discussing potential solutions. Collaborators will send or be sent messages in the course of satisfying responsibilities. CRC cards have emerged as a simple yet effective way to analyze scenarios. First proposed by Beck and Cunningham as a tool for teaching object-oriented programming (Beck and Cunningham, 1989), CRC cards have proven to be a useful development tool that facilitates brainstorming and enhances communication amongst developers.

Beck and Cunningham suggest driving a design towards completion with the aid of execution scenarios. We can start with only one or two obvious cards and start playing "what-if" scenarios. If the situation calls for a responsibility not already covered by one of the cards we either can add the responsibility to one of the classes, or create a new class to address that responsibility. Since objects in such a representation have responsibilities, these type of classes can capture a role class which is more functionally oriented than objects. As expressed in (Cain and Coplien, 1993), the basic abstraction in the model is *role*, a longstanding, stable locus of associated responsibilities in a process.

Path expressions (Campbell and Habermann, 1974), which originally have been derived from regular expressions to represent primitive operations such as sequence (a ; b), selection (a , b), and iteration a^+ or a^* , can be added to every CRC card to define a clear sequence of execution amongst responsibilities. The parallel primitive operation, (a || b), has also been included to represent concurrency between operations. As it is extremely difficult to characterise the possible behaviours of a role or a process through informal representations, we have used the formalism of path expressions to define the constraints on a role's behaviour.

Modelling Cooperation of Objects using Petri Nets

Petri nets have been used with great success in the modelling of concurrent and real-time systems (Murata, 1989; Peterson, 1981). Extending Petri nets to simulate processes has also the potential for increasing management's ability to understand and improve business processes (Peters and Schultz, 1993).

In recent years, Petri nets have been applied for the representation, simulation, and formal proving in many object-oriented research domains (Bastide et al., 1993; Battiston and Cindio, 1993; Becker and Moldt, 1993; Buchs et al., 1993). This has led to many different interpretations of Petri net primitives depending on the type of application when mapping them to real-world concepts. Such a variety suggests that a unique interpretation , at least in defining the primitive concepts, of Petri nets in terms of object-oriented concepts is necessary. The following simple interpretation of Petri nets primitives turns out to be useful for process modelling since it covers the most aspects of a process such as concurrency, coordination, and communication.

- Transitions: can be assumed to be methods or operations of objects
- Places: can represent states of an object, therefore, representing pre and post conditions before and after execution of a method (transition)
- Arcs: are messages exchanged between objects
- Tokens: can represent instances of objects (Peters and Schultz, 1993), however, their flow represents the flow of control which basically shows the enactment of a process through the activation of its components

Combining Petri nets with object-oriented concepts provides a good model for representing, understanding, and enacting the behaviour of roles and finally processes. This model is able to simulate the activation of different methods of different objects as firing transitions, messages as arcs, and different states of a role as places.

Every responsibility of a role listed in a CRC card is assumed to be modelled by a transition in a Petri net representation. Path expressions, are used to represent the constraints on the firing of such transitions. Viewing the sequence of execution of the responsibilities for a CRC card, left to right and top to bottom, models a thread of control within a role. Apart from interactions that should be specified wherever collaborations take place, the semantics of the thread of control for responsibilities within a CRC card is specified in terms of primitive operations of path expressions. It is been shown that for every path that has been defined by primitive operations, there is an equivalent Petri net which semantically represents the same meaning of such path (Lauer and Campbell, 1975). Figure 3 shows the equivalent Petri nets for such primitive operations, sequence, selection, iteration, and concurrency of path expressions. Basically, any path expression which can appear on a CRC card would be a combination of such primitive operations. Therefore, a path with no interaction can simply be represented by its equivalent Petri net which has the same semantics of thread of control in a role as its original semantics in terms of path expressions.

This, of course, only covers the semantics of responsibilities in terms of Petri nets where no interaction has been specified. Roles cannot be seen without their interactions, just as objects cannot be viewed in isolation. Indeed the importance of interactions for roles is the same as that of message passing for objects. However, the sake of simplicity, we specify the semantics of interactions separately. Thus, subsequently, when mapping the collaborators field on a CRC card to Petri nets, suitable semantics can be added to complete such representations.

An issue which influences the provision of precise semantics for interactions between roles is the synchronisation thread amongst such roles. This strictly defines the exit and return points of each role when drawing the Petri nets for interactions. However, such a classification only provides the required information to decide on the exit and return points when a collaboration takes place. Further, we need to decide how to represent every sort of interaction identified by its path expression. Depending on the path expression to which such interaction belongs, figures 4, 5, 6 and 7 will help to map an interaction to its equivalent Petri nets .

AN APPROACH TO APPLYING OBJECTS IN PROCESS MODELS

The following steps specify the approach and the reason for exploiting the concepts which have been defined in the previous section.

1. Capturing the roles of a process in terms of CRC cards

   Defining roles in terms of CRC cards is the starting point for our approach. As specified earlier, the characteristics of a CRC card contain all the necessary aspects of the role concept. At the first level of representing such roles as CRC cards, responsibilities can be drawn as the different steps which should be followed to accomplish the expected task of a role class. Although the granularity of such steps depends on the desired level of abstraction, in practice each step would be a meaningful action within a role which normally transforms the state of a role. Transformations can also be assumed to be associated with the methods of the objects which will be employed to play every step of that role. However, we do not need to think of objects and their participations at this stage. Collaborators will be other roles cooperating with the current role.

2. Adding path expressions to constrain the ordering of responsibilities

   By applying CRC cards, we have started from an informal definition of roles with imprecise semantics. Since the starting point should provide a precise yet abstract definition of roles to facilitate the following refinement steps, we need to specify the ordering of responsibilities (steps) which will be carried out to play a role. As it is extremely difficult to characterise the possible behaviours of a role or a process through the informal representations, we have used the path expressions to constrain a role's behaviour. As a role prescribes the way in which we need to use different objects, we should be able to define a pattern of such prescriptions in terms of primitive operations for a structured execution. When adding path expressions, we only view the responsibilities, since constraining the responsibilities does not cover the collaborators field in a CRC card.

3. Drawing Petri nets to represent the responsibilities within every CRC card

   Having added path expressions to CRC cards, we now have a precise representation of responsibilities for every role. However, to accomplish such responsibilities, a role needs to interact with other roles and therefore, we need to add the semantics of such interactions. Thus the next step defines the properties of the current step plus the new properties relating to collaborators. We have exploited Petri nets for this purpose because of their representational power. Using the corresponding Petri nets representation, Figure 3, a precise semantics of responsibilities for a CRC card can be provided.

4. Adding the corresponding Petri nets for collaborators

   We still need to clarify exactly what is meant by stating that every interaction between a class and its collaborators takes place through an interaction protocol. However, interaction semantics implicitly exist in path expressions and one needs only to explicitly represent such semantics. This can be done by providing a Petri net definition of synchronous, asynchronous, and rendezvous types of interactions between two roles. This step requires the addition of extra semantics to the previous definitions to make clear what is meant in every interaction. Using the corresponding Petri nets for interactions, figures 4, 5, 6, 7, a precise semantics of interactions for CRC cards can be provided.

5. Checking the process model by viewing the aggregation of all CRC cards

   Figure 8 shows the aggregation of CRC cards for the producer-consumer example after adding path expressions to the cards.

6. Simulating Petri net models (needs also initialisation of places)

   Figure 9 shows a Petri net model of the producer-consumer pattern example which has been extracted from its CRC cards through the above approach. The roles, producer, consumer, and buffer have been represented in separate columns to distinguish between them, each column showing one of such roles. In this way, we distinguish between internal arcs which show transformations from one state of a role to another state, and external arcs which show message exchange between two roles. Places between such arcs can be interpreted as representing the pre-conditions to a transition being fired. Using a Petri net tool to simulate the resulting model, after initialisation of places which is normally the initial state of a modelled process, will help the understanding and improve the expected behaviour of participant objects in a role and finally in a process.

These steps can be iterated until a satisfactory result is obtained. However, an important point is the finding of more collaborator objects by viewing every responsibility in detail.

APPLICATION OF THE APPROACH

Reuse, and simulation are two different applications of our approach and will be discussed briefly in this section.

The notion of reusability is as applicable to roles as it is to objects. We have encapsulated within a role its patterns of internal behaviour, and these can be ignored when reusing a role in a new process. The external interfaces alone characterise a role in terms of its reuse potential (Shams-Aliee and Warboys, 1995).

Good representation of roles facilitates their reuse. Figure 10 shows an encapsulated representation of different roles in the producer-consumer process example. The left facing arcs represent messages which a role needs to receive to be activated. The parallel arrow represents that a role should send a response to the client role. The right facing arcs represent messages which a role needs to send to other roles. The parallel arrow also shows that a role has to wait to receive the response from the server role.

Having provided a Petri net model of the process stating its agents in terms of roles, a Petri net tool can be used for simulation, thus achieving an enactment of a process model. Such a simulation enacts the model to aid understanding, checking, and improving the original specification of the process stated in terms of role interactions. Firing a transition shows the activation of a responsibility of a role which corresponds to an operation or method of an object. Viewing the movement of tokens through the different roles at the same time, simulates a multiple thread of execution within a process.

As roles are aligned in different columns, they have internal and external arcs to other components of a Petri net. Internal arcs which enable the movement of tokens through a role (a column) can represent the change of state within a role. Every incoming place to a transition represents a pre-condition which a role must fulfill to be able to perform (fire a transition) a specific step of such role. Every outgoing place also represents post conditions after firing a transition (after finishing a step) within a role. External connecting arcs, which provide interactions between roles, represent potential messages which are exchanged between roles. Those arcs which are input to a role can be viewed as an event for a role to allow the activation of a specific step of that role. Those arcs which are output from a role to other roles can be viewed as messages needing to be sent by a role to get help from other role to accomplish its task.

As an example, Petri nets drawn for the producer-consumer process can be simulated to represent the behaviour of the components of such a process. This can also show how different recognised components of a small process such as this may cooperate to achieve the goal of that process. Another feature of such a simulation allows for the possibility of studying the effects of a partial change in a process. For example, if we want to extend the capability of the represented model by increasing the number of its buffers, we need only increase the number of tokens when initialising tokens for the buffer. This approach is also applicable when increasing the number of tokens for producer or consumer roles which represent the existence of more producer or consumer in the process. Such an extension, showing the potential for reuse of roles involved in this process, demonstrates the suitability of the resulting model in meeting the aim of better reuse in process models.

CONCLUSIONS

The general intention of this research was to extend the applicability of the object-oriented concept from general software applications to process modelling applications. Consideration of possible solutions lead to another problem. That is the lack of a suitable intermediate level between the process and object world. Therefore, providing a bridge to convey the desirable advantages from the object world to the process world was the primary goal.

The approach presented in this work provides a basis for the refinement of roles in terms of their behaviour through the cooperation of different objects. This approach is useful since it can capture the responsibilities of a role at an abstract level. It also allows the adding of details to the abstract definition to identify the exact sequence of activities within a role.

CRC cards, as originally designed, assist in the task of the initial identification of objects. In this approach they can also assist in discovering the roles in a process. They seem to provide a good starting point for both refinement and brainstorming when identifying roles and their responsibilities.

This approach has also shown that by giving exact semantics to CRC cards, they become adequate for representing a suitable component of design, like a role in a process, since they cover all the necessary aspects such as encapsulation, communication, and the temporal behaviour of roles.

Providing a formal representation of the role concept in terms of object-oriented Petri nets presents a clear semantics for this concept helping to add formality to the whole process model. Therefore, applying Petri nets has resulted in a simple yet formal and enactable model of a process.

Finally, viewing the design of process models in a similar way to the design of general software applications increases the adoptability of the object-oriented view as a general modelling approach since it has been applied successfully (Jacobson et al., 1995) to a new application area, in this case process technology.

References

Andersen, E. P. and Reenskaug, T. (1992). System Design by Composing Structures of Interacting Objects. In *ECOOP'92 Proceedings, LNCS 615*, pages 133–152. Springer-Verlag.

Arapis, C. (1990). Specifying Object Life-Cycles. In Tsichritzis, E. D., editor, *Object Composition*, pages 197–225. University de Geneve.

Baldi, M., Gai, S., Jaccheri, M., and Lago, P. (1993). An Initial Experiment with Object-Oriented Software Process Modelling. Department of Automation and Informatics, Polytechnic of Torino, Italy.

Bastide, R., Sibertin-Blanc, C., and Palandue, P. (1993). Cooperative Objects: A concurrent, Petri-Net Based, Object-Oriented Language. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vol. 3*, pages 286–291.

Battiston, E. and Cindio, F. D. (1993). Class Orientation and Inheritance in Modular Algebraic Nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vol. 2*, pages 712–723.

Beck, K. and Cunningham, W. (1989). A Laboratory For Teaching Object-Oriented Thinking. In *OOPSLA'89 Proceedings*, pages 1–6. ACM SIGPLAN Notices, vol. 24, no. 10.

Becker, U. and Moldt, D. (1993). Object-Oriented Concepts for Coloured Petri Nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vol. 3*, pages 279–285.

Belkhatir, N. and Melo, W. L. (1994). Tempo: Defining Software Processes in an Approach Based on Objects with Roles. In *In Proceedings of the 1st International Conference on Object-Role Modelling*, Magnetic Island.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications.* The Benjamin/Cummings Publishing Company Second Edition.

Buchs, D., Flumet, J., and Racloz, P. (1993). SANDS/CO-OPN Structured Algebraic Net Development System Concurrent Object-Oriented Petri Nets. Technical Report 71, Universite de Geneve.

Buschmann, F. (1993). Rational Architecture for Object-Oriented Software Systems. *JOOP September*, pages 30–41.

Cain, B. G. and Coplien, J. O. (1993). A Role-Based Empirical Process Modelling Environment. In *ICSP2 Proceedings*, pages 125–133, Berlin, Germany.

Campbell, R. H. and Habermann, A. N. (1974). The Specification of Process Synchronization by Path Expressions. In *Lecture Notes in Computer Science*, volume 16, pages 89–102. Springer Verlag.

Conradi, R. and et al (1994). *EPOSE: Object-Oriented Cooperative Process Modelling*, chapter three in Software process Modelling and Technology, Ed. Anthony Finkelstein and Jeff Kramer and Bashar Nuseibeh, pages 33–70. Research Studies Press LTD.

Curtis, B., Kellner, M., and Over, J. (1992). Process Modeling. *Communications of ACM*, 35(9):75–90.

Eliens, A. (1995). *Principles of Object-Oriented Software Development.* Addison-Wesley.

Garg, P. K., Mi, P., and Pham, T. (1994). The SMART Approach for Software Process Engineering. In *In Proceedings of the 16th International conference on Software Engineering*, pages 341–350, Italy. IEEE.

Jacobson, I., Ericsson, M., and Jacobson, A. (1995). *The Object Advantage: Business Process Reengineering with Object Technology.* Addison Wesley.

Johnson, R. E. (1992). Documenting Frameworks using Patterns. In *OOPSLA '92 Proceedings*, pages 63–76. ACM SIGPLAN Notices, vol. 27, no. 10.

Lauer, P. E. and Campbell, R. H. (1975). A Description of the Path Expressions by Petri Nets. In *Proceedings of the 2nd ACM Symposium on Principles of Programming Languages*, pages 95–105, New York, ACM.

Milner, R. (1989). *Communication and Concurrency.* Prentice-Hall.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580.

Osterweil, L. (1987). Software Processes are Software too. In *Proceedings of the 9th International Conference on Software Engineering*, pages 2–13, Monterey.

Peters, L. and Schultz, R. (1993). The Application of Petri-Nets in Object-Oriented Enterprise Simulations. In *Proceedings of the 26th Hawaii International Conference on System Science, Vol. 4*, pages 390–398. IEEE.

Peterson, J. L. (1981). *Petri Net Theory and the Modelling of Systems.* Prentice Hall.

Reenskauge, T. and et al (1992). OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems. *JOOP October*, pages 27–41.

Rumbaugh, J., Blaha, M., Premelani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modelling and Design.* Prentice-Hall.

Shams-Aliee, F. and Warboys, B. C. (1995). Roles Represent Patterns. In *the Workshop on Pattern Languages of Object-Oriented Programs at ECOOP'95*, Aarhus University, Denmark.

Singh, B. and Rein, G. L. (1992). Role Interaction Nets (RINs): A Process Description Formalism. Technical Report CT-083-92, MCC.

Warboys, B. (1989). The IPSE 2.5 Project: Process Modelling as a basis for a support environment. In *Proceedings of the First International Conference on Software Development, Environments, and Factories*, Berlin.

Wiernga, R. and de Jonge, W. (1993). The Identification of Objects and Roles. Technical Report TR-267, Faculty of Mathematics and Computer Science, Vrije University.

Wirfs-Brock, R. J. and Johnson, R. E. (1990). A Survey of Current Research in Object-Oriented Design. *Communication of ACM*, 33(9):104–124.

Wirfs-Brock, R. J. and Wilkerson, B. (1989). Object-Oriented Design: A Responsibility-Driven Approach. In *OOPSLA'89 Proceedings*, pages 71–75.

Wirfs-Brock, R. J., Wilkerson, B., and Wiener, L. (1990). *Designing Object-Oriented Software.* Prentice Hall.

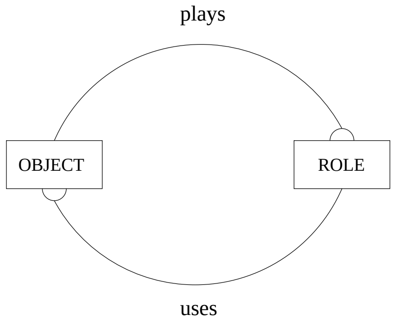

Figure 1: E-R Model of Object and Role Concepts

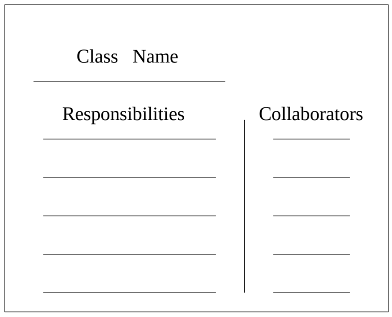

Figure 2: A Class, Responsibilities, and Collaborators (CRC) Index Card

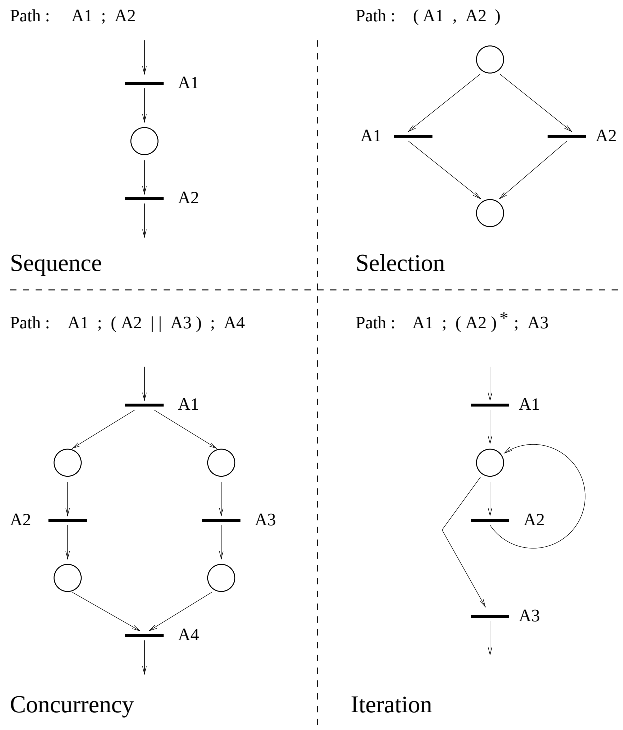

Figure 3: Equivalent Petri nets for Primitive Operations in Path Expressions

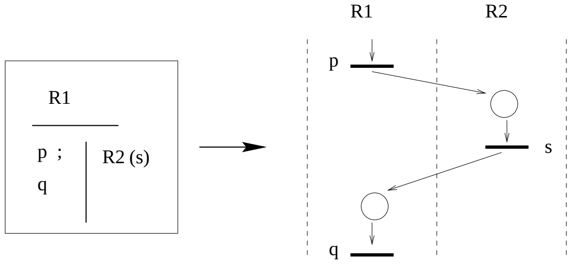

Figure 4: Equivalent Petri nets for Sequential Operations in CRC

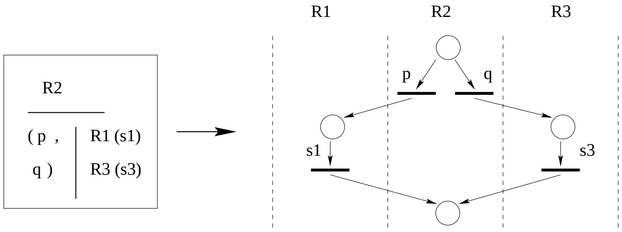

Figure 5: Equivalent Petri nets for Selection Operations in CRC

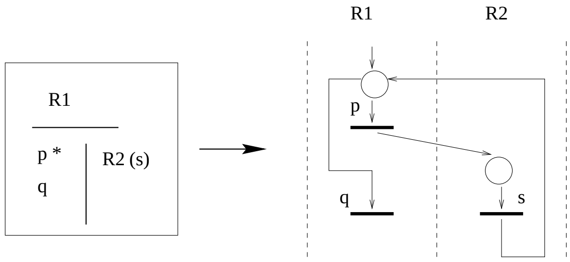

Figure 6: Equivalent Petri nets for Iteration Operations in CRC

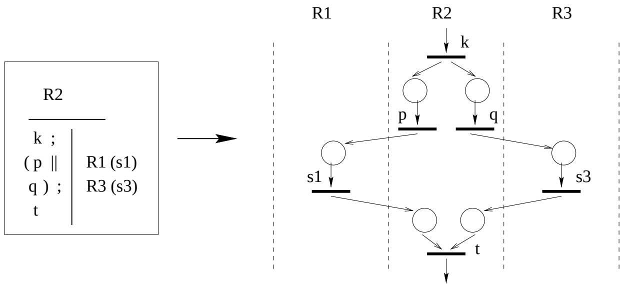

Figure 7: Equivalent Petri nets for Concurrent Operations in CRC

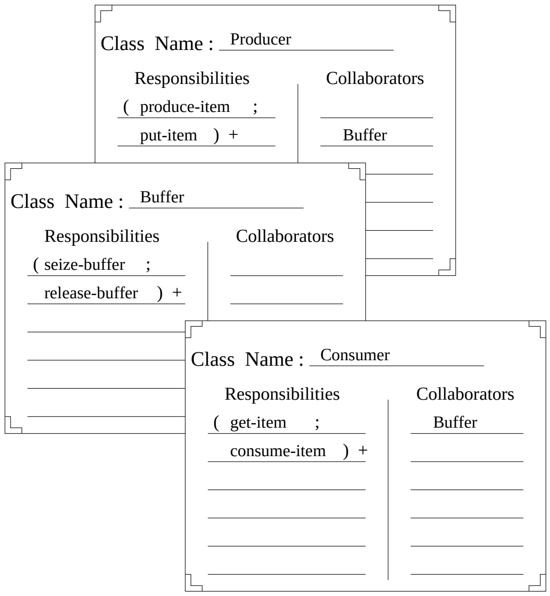

Figure 8: The CRC Cards for Producer-Consumer Example

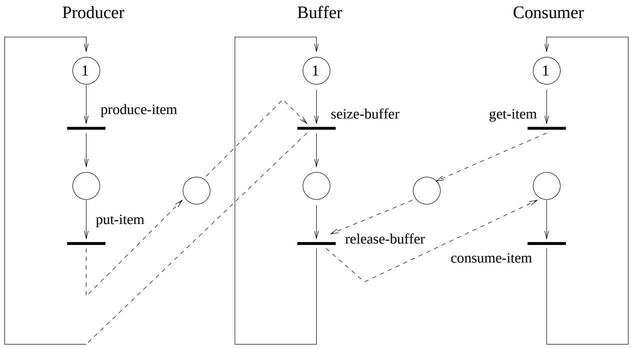

Figure 9: The Petri nets for Producer-Consumer Example

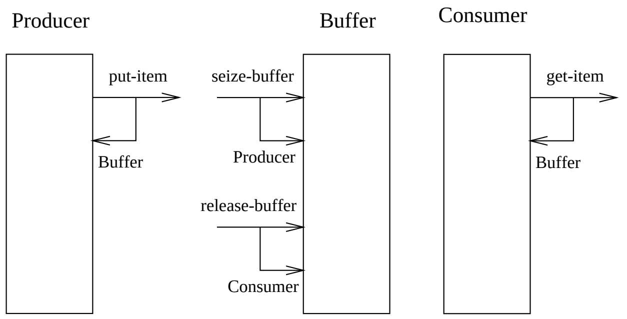

Figure 10: An Encapsulated Representation of Roles with their Interactions