Roles Represent Patterns

Fereidoon Shams Aliee   Brian C. Warboys

Department of Computer Science,
University of Manchester,
Manchester M13 9PL,
U.K.
E-mails : ferei@cs.man.ac.uk   brian@cs.man.ac.uk

Abstract

In building process models based on objects, we identify a gap between object capabilities and the process world. To overcome this problem, a level of abstraction, role, above the object level, similar to pattern structure, has been defined. We have represented roles in different levels of abstraction containing CRC cards, formalised CRC (CRC plus path expressions), Petri nets, and encapsulated structures to provide a pattern representation of roles.

Since it is important to understand a pattern before reusing it, a representation method to encapsulate a solution for a pattern based on roles which rely on cooperative objects is suggested. An approach to refine the representations also helps to specify the exact semantics of such representations.

1 Introduction

The primary motivation for this research was experimentation with the design of process models based on the collaboration of cooperative objects. When exploiting objects for this purpose, we have treated process models like other software application domains. The reason perhaps is the adoption of the idea that *software processes are software too* [16], and of similarities between designing software and designing process models. Such models basically represent the behaviour of different objects involved in the process.

When representing a model of a process based on the cooperation of different objects, it is difficult to design a process model solely in terms of such objects. We observed that by defining a level of abstraction above the object level we can remedy this situation. Moreover, we noted that there is a general problem in designing many other object-oriented applications in that is the need to identify another level to improve reusability and facilitate design [13, 12, 6, 21, 17]. This problem can be addressed by defining a new level of abstraction, above the basic object concept level, for modelling a group of objects which cooperate with each other to accomplish a task as a process component.

Recently, the importance of providing a framework of collaborating objects to improve reusability has been recognised by many experts in the object-oriented design field. Frameworks [13], patterns [11], collaborating graphs [22], role teams [6], mechanisms [5], contracts [12], role models [17] are some examples of such efforts. The main focus has been to identify, capture, and use an abstraction level above the concept of object to provide a better representation of software architecture which consequently can improve the quality of software design.

Similarly, when modelling a process based on communicative objects, recognising a framework to classify collaborating objects seems to be clearly useful. Such a framework, as the basic abstraction in a process model, has many similarities with the concept of *role* which can be viewed as a list of responsibilities assigned to an agent and needing to interact with other objects to accomplish its task within a process. Therefore, role modelling is now associated with the concept of collaborating objects to provide a high level abstraction to improve the application of object-oriented modelling [17, 10].

Designing patterns has also such an intention and we have found many similarities between the notion that we have defined as the abstraction level above objects and the notion of pattern. Most of the suggested pattern definitions are based on traditional class relationships. However, we find it difficult to represent patterns only based on class relationships of objects, since this only provides a static view of such representations. A dynamic view point, which is most desirable in modelling processes based on objects, is the representation of role-interaction relationships between objects to specify a pattern. Therefore, the focus of this work is to define an intermediate level between process and object levels. By creating such a definition, we are a step nearer the concept of pattern and yet still retain the original notion of object in the new level.

On the other hand, it seems that in designing patterns, most attention has been paid to recognising and defining these patterns in different applications. While representation of such patterns in an understandable form, to be reused by different people, has been given less attention. It seems clearly important to provide a high level representation method to facilitate encapsulating, understanding, and reusing patterns. If we are going to use the notion of pattern at the design level, we have to be able to represent our patterns in terms of design representations rather than programming languages.

In this paper, we specify the notion of role and its relation to the object concept. Then we discuss how roles can represent patterns. An approach demonstrating the usefulness of defining such patterns and the way in which to extract such patterns for designing process models is presented. Finally, a representation method which covers different levels of abstraction and which is suitable for encapsulating patterns in terms of roles is specified.

2 Roles and Objects

Role definition is basically concerned with role theory as used for organisational coordination. Here, roles are the building blocks of organisational structures [18]. 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 in 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 by Booch [5], a role is the face which an object presents to the world at a given moment or a particular aspect or behaviour which an object

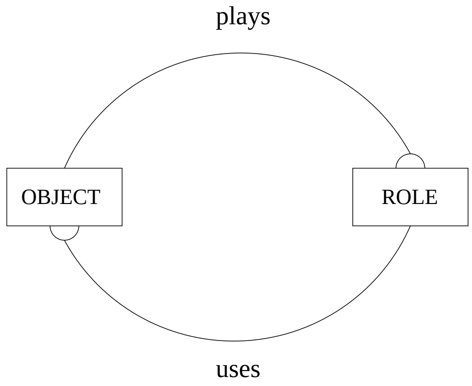

Figure 1: E-R Model of Object and Role Concepts

exhibits during a period of time [2]. 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 these 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 other objects can be used.

In designing process models, roles are the basic building blocks of processes [19]. In contrast with other software applications, role can be seen as a small application module 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 roles. For example, depending on the level of detail that we wish to model, some objects which cooperate in the playing of the project manager's role are project manager as a person object, documents, files, E-mail tool, and so on.

In [20] a precise distinction between role and object concepts has been made and some good examples were 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 a component of a process based on its behaviour in terms of the cooperation between one or several objects in playing that role. This also prevents the generation of new technical problems, due to complexity, when trying to use objects directly for modelling processes.

By applying objects to play a role, the notion of role can gain 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.

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 [15]. Recognising a pattern initially by its roles is simpler than directly viewing that pattern in terms of its objects.

3 Roles can help to Identify Patterns

The *role* concept is very close to the concept of *pattern* in the object-oriented design field, since it represents patterns of objects by prescribing the way objects cooperate in order to accomplish a specific task. Further when describing a process based on the role concept, it provides a richer vocabulary than pure object concepts alone.

Roles represent patterns as they can be used to build a process architecture. However, the way roles cooperate within a process is slightly different from object cooperations. This difference should be addressed by defining an abstract interface for role interactions in a formal way.

We have found some similarities when recognising roles as patterns as follows:

- Patterns and roles both represent a solution to a recurring problem in terms of cooperative objects
- They both are a high level of abstraction which is suitable for application at the design level
- Both need to be well captured, classified, and represented for reuse

Providing the possibility of defining more roles in a software application means the possibility of defining new patterns for that software application. Roles can repeat in different processes (applications) as patterns are expected to and role models represent the structure of processes as patterns generate software architectures [4].

4 Characterising Roles

Class, Responsibilities, Collaborators (CRC) cards were invented by Cunningham in response to a need to document collaborative design decisions [3]. It is recognised as a useful method for building structures of communicating objects (roles) in many object-oriented applications [1, 7, 8, 17, 22]. CRC definitions are very close to capturing the nature of a role because they have been defined using a responsibility-driven approach rather than a data-driven approach [21].

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 of analyzing scenarios. First proposed by Beck and Cunningham as a tool for teaching object-oriented programming [3], 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 toward 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 the 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

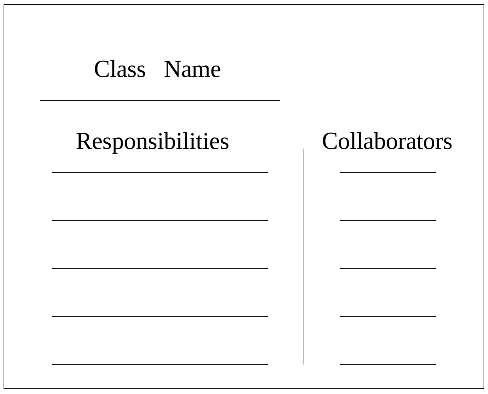

Figure 2: A Class, Responsibilities, and Collaborators (CRC) Index Card

[8], the basic abstraction in the process model is *role*, a longstanding, stable locus of associated responsibilities in a process.

Therefore the notions of class, responsibilities, and collaborators in a CRC card may have the existing properties of objects plus some richer properties when used to identify roles as described below:

- Role class: More abstract than object class and can be refined to several cooperative objects
- Responsibilities: More dependent than object's responsibilities as there is usually a functional cohesion between responsibilities within a role
- Collaborators: They are not only the servers but also the partners in teamwork. Therefore, we refer to collaborators as any kind of necessary interaction amongst roles

5 A Refinement Approach to Discover the Structure of Patterns

We have developed an approach which formalises CRC cards to represent all the necessary aspects of the notion of role.

The aim of this approach is to provide a basis for the refinement of a unit of design for a process model, called a *role* here, towards representing the patterns of behaviour of participant objects and their collaborators in terms of Petri nets. This approach is based on constraining the order of responsibilities within a CRC card.

Path expressions [9], 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.

Figure 3. shows the CRC cards for a producer-consumer pattern example after adding path expressions to these cards. As a role prescribes the way in which we need to use

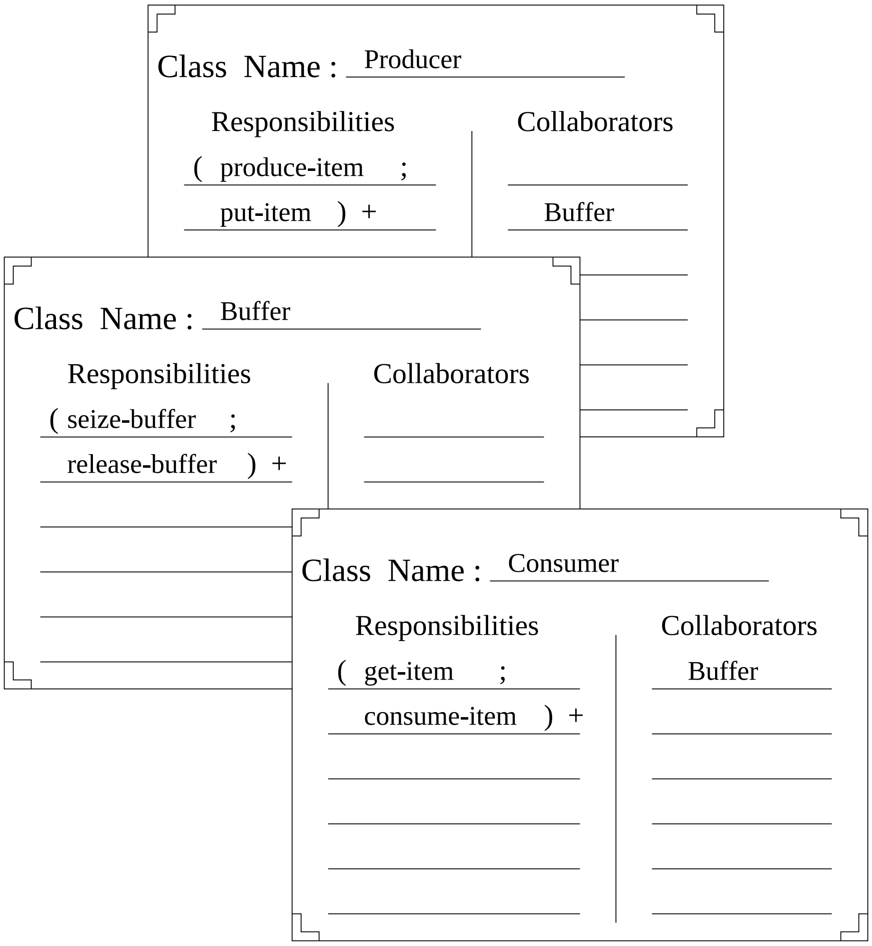

Figure 3: The CRC Cards for Producer-Consumer Pattern Example

different objects, we should be able to define a pattern of such prescriptions in terms of primitive operations for a structured execution. This will help clarify what should be done in terms of primitive operations such as sequence, alternation, and iteration of activities (responsibilities). However, we still need to clarify what we mean by stating that every interaction between a class and its collaborators takes place through an interaction protocol. 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. Combining Petri nets with object-oriented concepts provides a good model for representing, understanding, and enacting the behaviour of roles and finally processes.

We can summarise the approach as the following:

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

2. Adding path expressions to constrain the ordering of responsibilities

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

4. Adding corresponded Petri nets for collaborators

5. Finding more collaborator objects by viewing every responsibility in detail

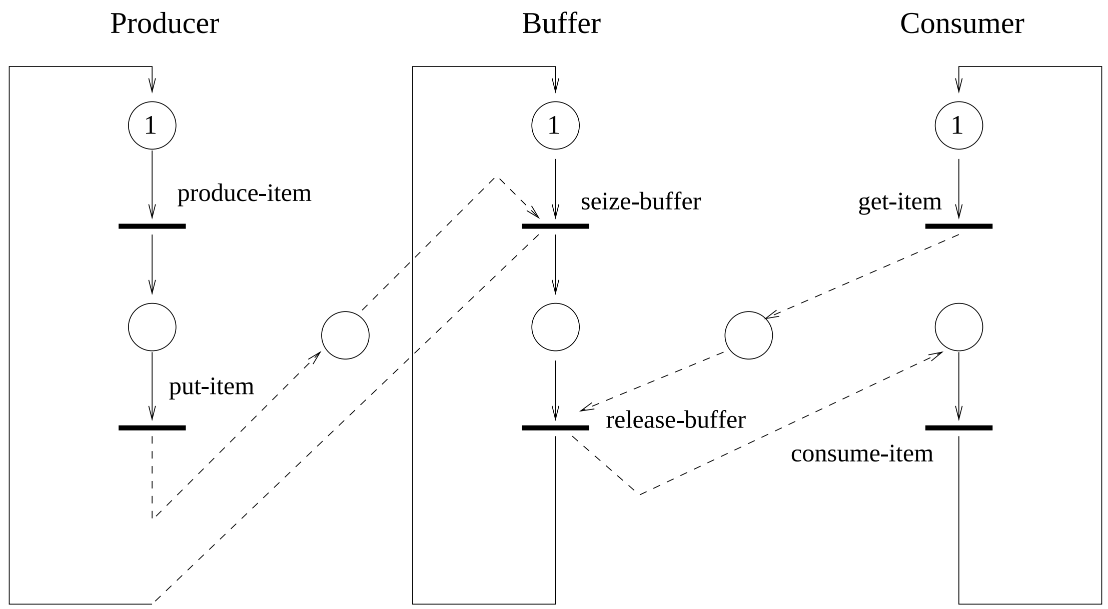

Figure 4: The Petri-Nets for Producer-Consumer Pattern Example

6. Checking the process model by viewing the aggregation of all CRC cards
7. Simulating Petri net models (needs also initialisation of places)

These steps can be iterated until a satisfactory result is obtained.

Figure 4. shows a Petri net model of the producer-consumer pattern example which has been extracted from its CRC cards using this approach.

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 events for a role allowing 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 another role to accomplish the 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 is that it 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.

6 A Representation Method to Reuse Patterns of Roles

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. Therefore, the notion of collaborators in CRC cards when applied to roles will broadly cover any sort of interaction which can take place between two roles. We can classify interactions in a CRC card to the following types:

- Asynchronous

  1. A role needs to receive a message from its collaborator
  2. A role should send a message to its collaborator(s)

  Note that they are asynchronous interactions since a caller does not need to wait after sending the expected message.

- Synchronous

  1. A role should send a message to its collaborator and has to wait until receiving the response
  2. A role needs the collaboration of some other role(s) at a specific time such as a conversation or a teamwork; so called rendezvous

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.

We have used such a classification to distinguish the type of external interactions between roles. The separation of internal behaviour from external interfaces makes the role concept recognizable as a well defined component in different process models.

Good representation of patterns facilitates their reuse. This also is true for roles. Figure 5. shows an encapsulated representation of different roles in the producer-consumer 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 others roles. The parallel arrow also shows that a role has to wait to receive the response from the sever role.

Figure 6. shows another level of abstraction which represent the structure of a pattern in terms of its roles. This can also be seen as a solution for producer-consumer pattern as a reusable component for other design applications.

A more concrete example of the approach representing such pattern structures is included in the appendix of this paper.

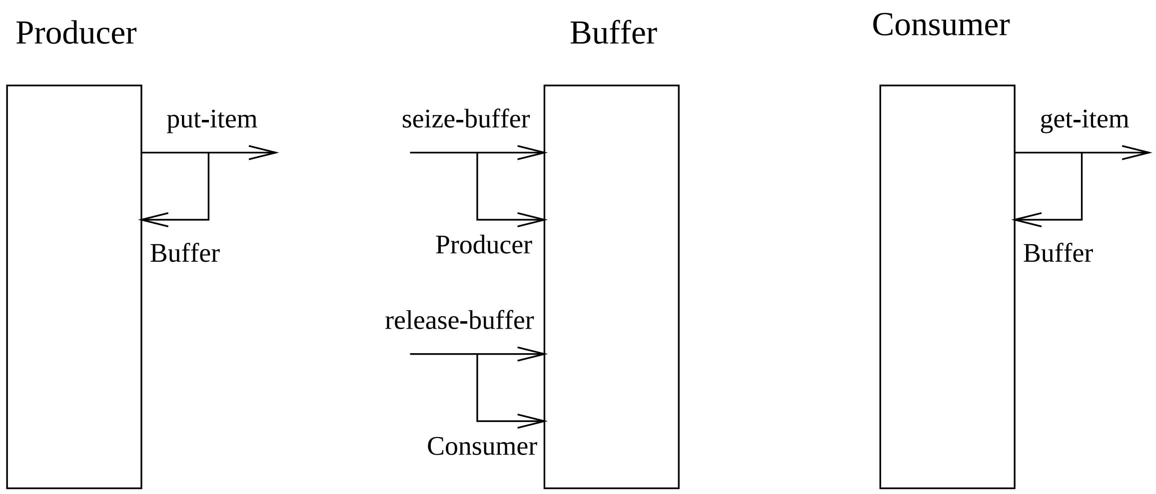

Figure 5: An Encapsulated Representation of Roles with their Interactions

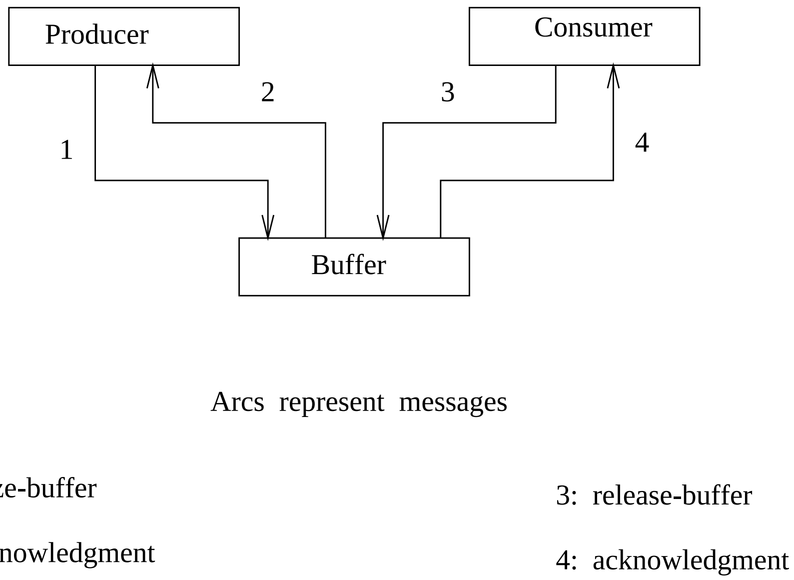

Figure 6: General Structure of Producer-Consumer Pattern

7 Conclusions

Defining a meaningful level of abstraction for both the object and the process world can facilitate the application of object-oriented concepts in process modelling and also provides a good understanding of both the object and the process world.

The approach presented in this work provides a basis for 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 and also allows for the adding of more details to the abstract definition to discover 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.

The importance of encapsulating and representing patterns is no less than finding such patterns. However, finding patterns is usually limited to a domain while representing patterns is an issue of all application's interest. In this research, the representation of patterns based on class relationships has been replaced by role interaction relationship. Based on such structures, we can recognise which messages externally need to be exchanged between roles. Providing a structured view of roles in terms of their external interactions represents a configuration of roles, based on the cooperation of objects, that solve a recurring problem. The structured view of roles therefore represents a pattern.

References

[1] Egil P. Andersen and Trygve Reenskaug. System Design by Composing Structures of Interacting Objects. In *ECOOP'92 Proceedings*, 1992.

[2] Constantin Arapis. Specifying Object Life-Cycles. In Ed. Dennis Tsichritzis, editor, *Object Composition*, pages 197–225. University de Geneve, 1990.

[3] Kent Beck and Ward Cunningham. A Laboratory For Teaching Object-Oriented Thinking. In *OOPSLA'89 Proceedings*, 1989.

[4] Kent Beck and Ralph Johnson. Patterns Generate Architectures. In *ECOOP'94 Proceedings*, 1994.

[5] Grady Booch. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company Second Edition, 1994.

[6] R.J.A. Buhr and R.S. Casselman. Timethread-Role Maps for Object-Oriented design of Real-Time and Distributed Systems. In *OOPSLA' 94 Proceedings*, 1994.

[7] Frank Buschmann. Rational Architecture for Object-Oriented Software Systems. *JOOP*, September 1993.

[8] B. G. Cain and J. O. Coplien. A Role-Based Empirical Process Modelling Environment. In *ICSP2*, Feb. 1993.

[9] R. H. Campbell and A. N. Habermann. The Specification of Process Synchronization by Path Expressions. In *Lecture Notes in Computer Science*, volume 16. Springer Verlag, 1974.

[10] Ronald S. Casselman. A Role-Based Architecture Model Applied to Object-Oriented Systems. Master's thesis, Department of Systems and Computer Engineering, Carleton University, 1993.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In *ECOOP '93 Proceedings*, 1993.

[12] Richard Helm, Ian M. Holland, and Dipayan Gangopadhyay. Contracts: Specifying Behavioral Compositions in Object-Oriented Systems. In *ECOOP/OOPSLA '90 Proceedings*, 1990.

[13] Ralph E. Johnson. Documenting Frameworks using Patterns. In *OOPSLA '92 Proceedings*, 1992.

[14] M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H.D.Rombach. ISPW-6 Software Process Example. In *Proceedings of the First International Conference on Software Process*, Washington, 1991. IEEE Computer Society Press.

[15] Doug Lea. Roles Before Objects. Technical Report uncompleted, Computer Science Department, State University of New York at Oswege, 1995.

[16] L.J. Osterweil. Software Processes are Software too. In *Proceedings of the 9th International Conference on Software Engineering*, Monterey, 1987.

[17] Trygve Reenskauge and etal. OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems. *JOOP*, October 1992.

[18] Baldev Singh and Gail L. Rein. Role Interaction Nets (RINs): A Process Description Formalism. Technical Report CT-083-92, MCC, 1992.

[19] B.C. Warboys. 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, 1989.

[20] Roel Wiernga and Wiebren de Jonge. The Identification of Objects and Roles. Technical Report TR-267, Faculty of Mathematics and Computer Science, Vrije University, 1993.

[21] Rebecca J. Wirfs-Brock and Brian Wilkerson. Object-Oriented Design: A Responsibility-Driven Approach . In *OOPSLA'89 Proceedings*, 1989.

[22] Rebecca J. Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. *Designing Object-Oriented Software*. Prentice Hall, 1990.

8 Appendix : A Fragmentation of the ISPW-6 Example

At the 6th International Software Process Workshop (ISPW-6), an example software process modelling problem [14] was designed to aid understanding and for comparing various approaches to software process modelling. This definition is scoped as a relatively confined portion of the software change process. It focuses on the designing, coding, unit testing and management of a rather localised change to a software system. This is prompted by a change in requirements. It is assumed that the proposed requirements change has already been analyzed and approved by the Configuration Control Board (CCB). It has also been determined that only a single code unit is to be affected. The process begins with the project manager scheduling the change and assigning the work to appropriate staff (here we assumed they are design engineer and QA engineer). The process ends when the new version of the code has successfully passed the new tests.

Three different roles have been recognised for this process, *Project Manager, Design Engineer,* and *QA Engineer* as the most obvious agents of such an example. Although at a detailed level of the process one might recognise some other role instances, for the sake of brevity and the fitting of such a big example into the framework of this approach, we have taken a high level interpretation of the example.

Figure 7. shows the CRC cards for the ISPW-6 example. Such a CRC description of this example, however, clarifies a subset of the original ISPW-6 definition [14] as our interpretation of the example which will be examined under the approach. Since there might be more one role responsible for the playing of a particular step, such as reviewing design, which both design and QA engineers are responsible for, such responsibility has been shown in two parts: e.g. Review Design(1) and Review Design(2). Further, we assume that only one of these roles is responsible for being questioned about that step and reporting the results to the authorities, if required. Therefore, as shown in related CRC cards, in playing the Review Design, Design Engineer is responsible and will rely on services from QA Engineer as its collaborator. Also in playing Test Unit, QA Engineer is recognised as responsible and will rely on services from the Design Engineer. It is also assumed that the scheduling and assigning tasks should rely on the collaboration of other participant roles, since while such roles have not accepted new tasks, this step cannot be considered as being completed.

As figure 7. shows, it has clarified that the Monitor Progress needs to be repeated as many times as required. Also Modify Design and Review Design should be allowed to repeat until a satisfactory result is obtained. Based on the example definition, QA Engineer can start reviewing design as soon as the task is assigned. Therefore, the concurrency of performing its other responsibilities with Review Design is represented by its related operation in path expressions.

Figure 8. shows an extracted Petri net model, semantically identical to its CRC model, for ISPW-6 example. Figure 9. shows an encapsulated representation of roles within ISPW-6 process example to represent the cooperation between roles. Figure 10. shows the general structure of ISPW-6 as a pattern example representing the interactions between the involved roles.

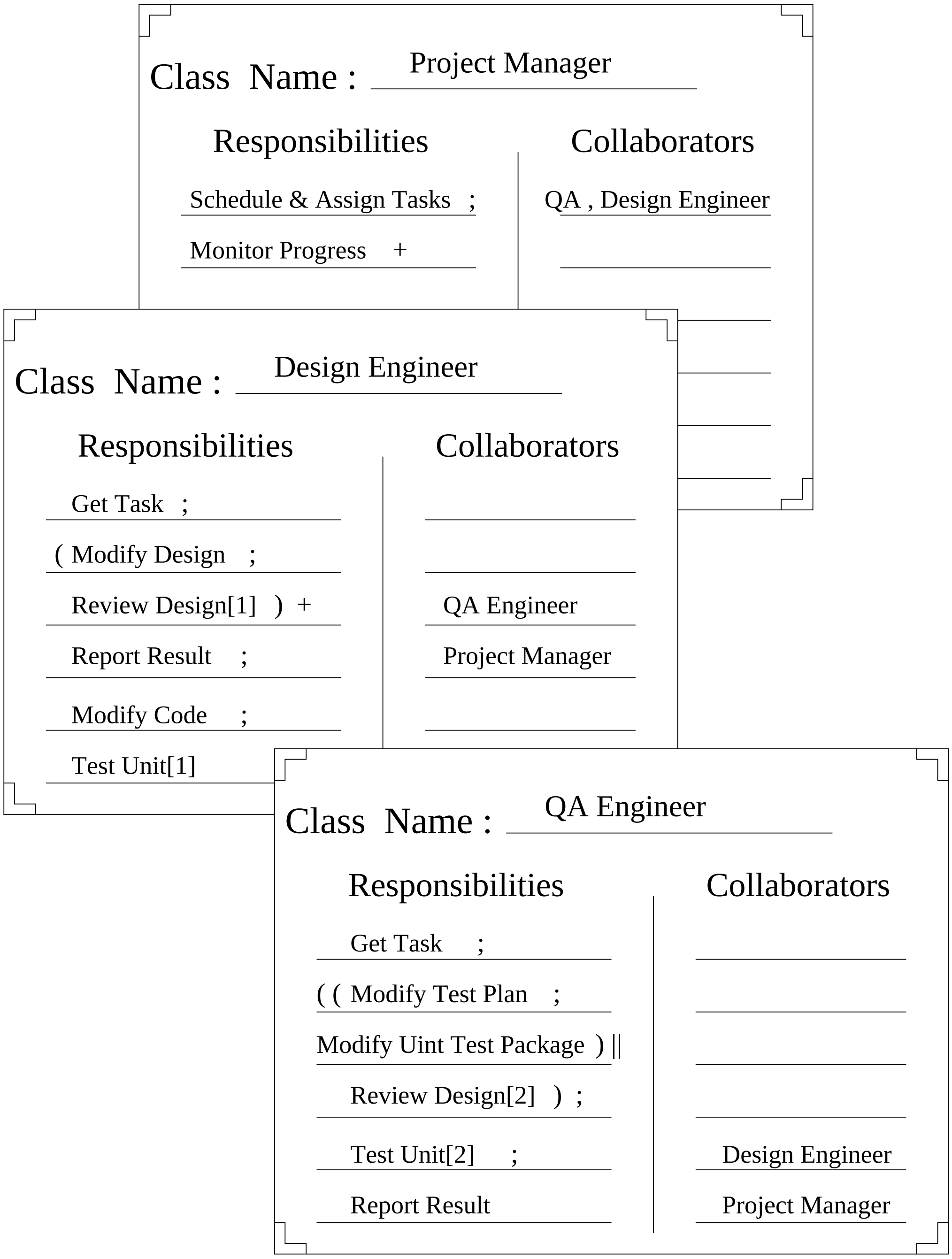

Figure 7: The Formalised CRC Cards for ISPW-6 Process Example

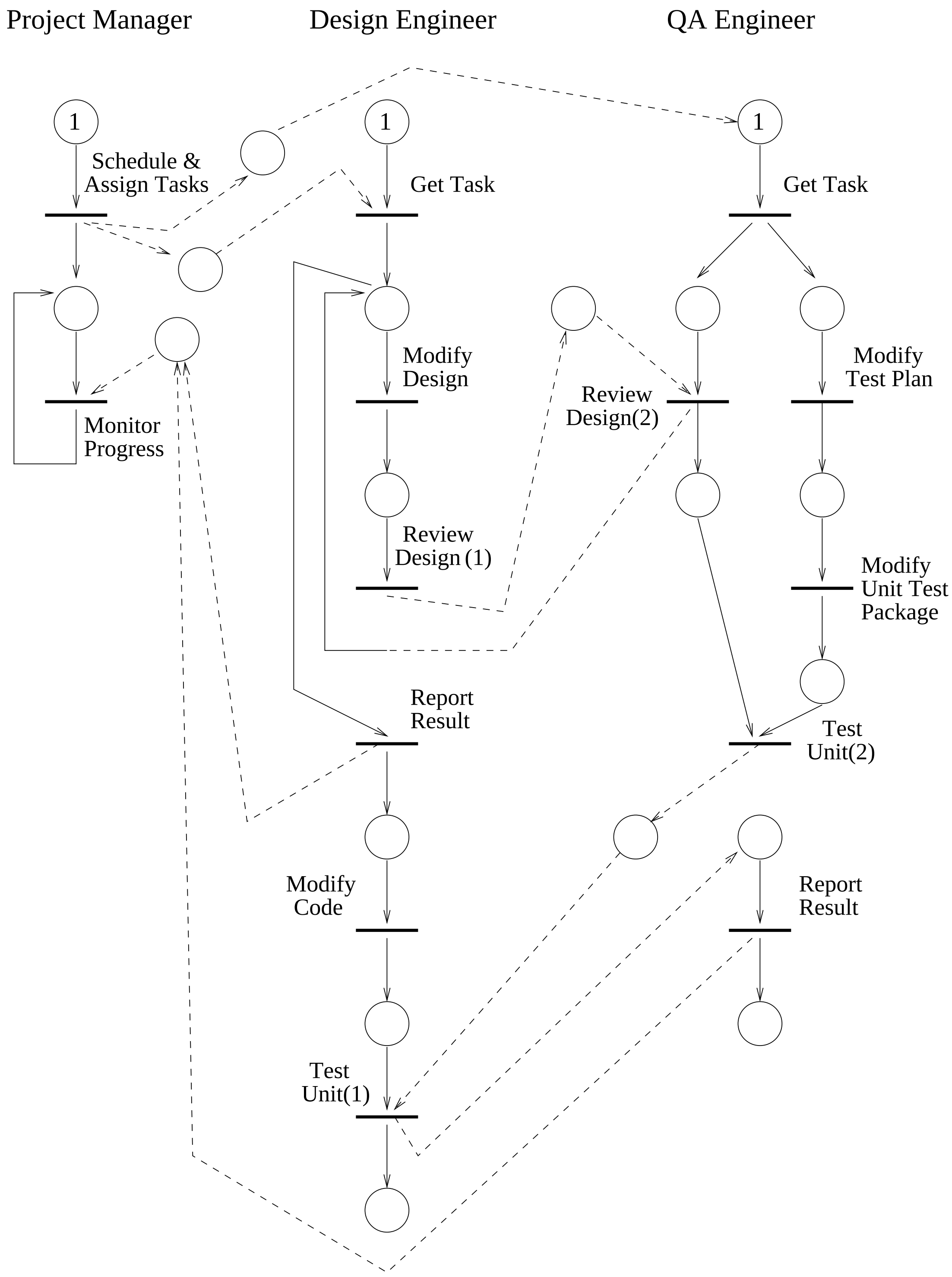

Figure 8: The Extracted Petri Nets for ISPW-6 Process Example

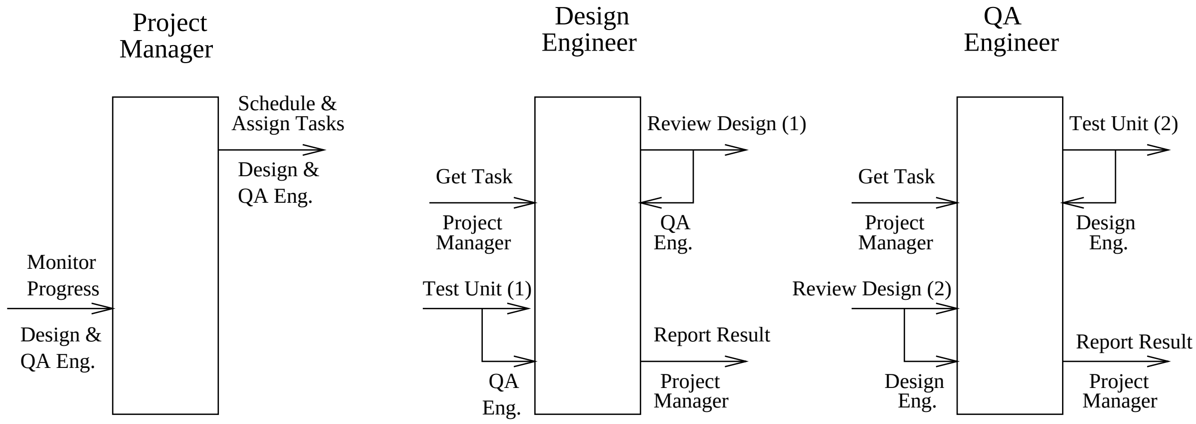

Figure 9: An Encapsulated Representation of Roles for ISPW-6 Example

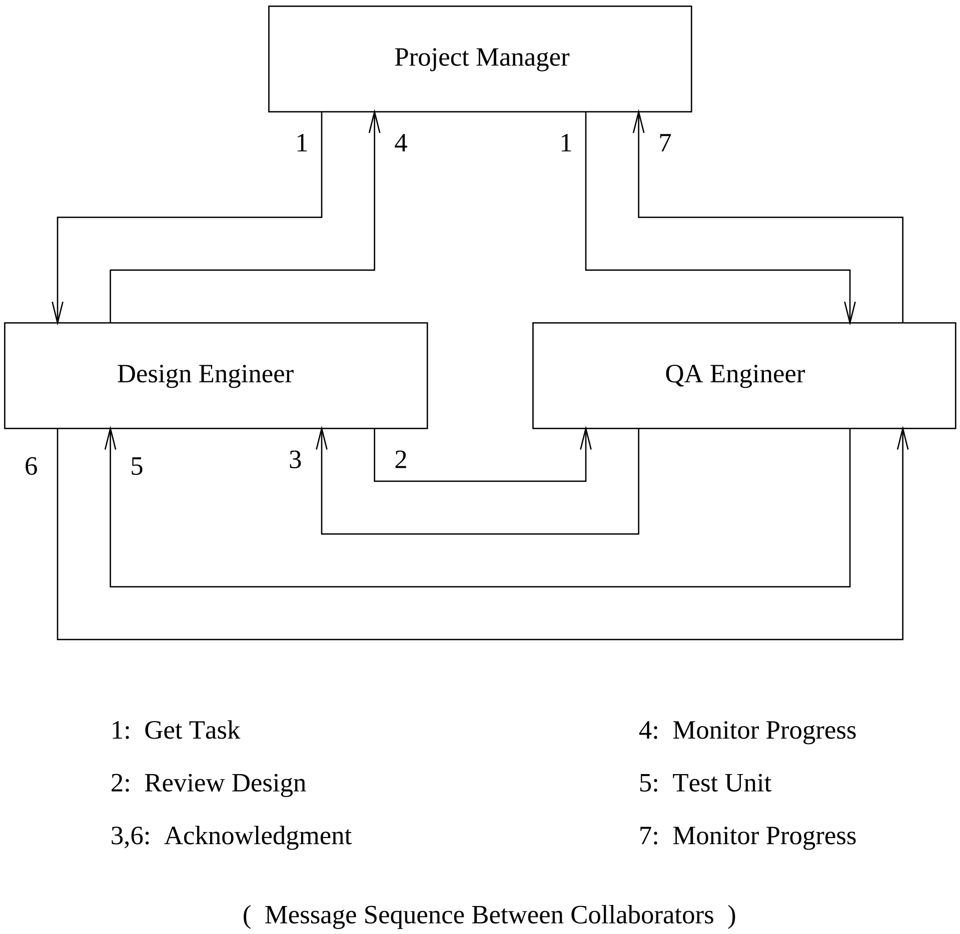

Figure 10: The General Structure of ISPW-6 Pattern Example