IJCES International Journal of Computer Engineering Science , Volume1 Issue 3, December 2011
ISSN : 2250:3439
https://sites.google.com/site/ijcesjournal
http://www.ijces.com/

Representing System Processes using the Actor Model / Processor Net

Anthony Spiteri Staines

Department of Computer Information Systems, Faculty of ICT,
University of Malta, Msida MSD 2080, Malta

Abstract. This paper describes the issue that modern systems, being composed of multiple components, have certain processing requirements that need to be properly addressed. Unfortunately very few models and notations provide for the idea of process modeling. For this purpose, separate notations have to be used. This work proposes the use of notations that have a specific focus on process modeling concepts. The solution is to use the Actor Model/ Processor Net. The solution is taken a step further by suggesting the use of Processor Network Patterns. These can be useful for describing and categorizing typical behavior in different types of systems.

Keywords: Actor Model, Design Patterns, Process Modeling, Processor Net

1 Introduction

Given systems complexity issues, many different notations and techniques have been developed for modeling systems. The main focus of these techniques separates the behavioral aspects from the static ones. There exists no perfect notation for modeling the behavioral aspect. The problem is made worse if proper process modeling is considered. Traditionally, process modeling is represented using special techniques or notations. When processes are modeled, the activity or process is shown separate from the entity causing the process to occur. Both the actor and the process are fundamentally interlinked.

Process modeling is of fundamental importance to different types of information systems, hardware modeling, computer networking and even systems like manufacturing systems.

2 Background and Motivation

Consider any system. The main system parts can be described using different diagrams, models or languages. Some might be visual whilst others can be more symbolic or mathematical. The obvious building blocks, consisting of components or main components, are normally depicted graphically. For this a variety of techniques exist, such as those in object oriented modeling, modeling languages and formal methods.

The idea here is not to create a new method or suppress the use of others but to use diagrammatic notations that simplify possible descriptions making it more modular and structured with the ability to combine blocks and create new formations, whilst simultaneously use and extract possible high level Petri nets from the structure. Further analysis could be applied as required. At the same time, the models must be more modular and comprehensible than current techniques in use. The idea of compactness is important for successful application which is not normally seen in Petri net structures [5]-[6]. The approach for construction has to be more intuitive. Components or sub-components need to connect and communicate properly. Interactions between different entities need to be properly understood. The ability to express system behavior is important for system architects, developers, project managers and different system stakeholders. Evidence of this is in large scale industrial system projects. Before even designing systems the interactions related to system processes have to be identified and understood. These ideas are clearly visible in FMC (Fundamental Modeling Concept) techniques, design patterns, MDE(Model Driven Engineering), SOA (Service Oriented Architectures), etc. The idea of abstracting is to comprehend systems even though there are great differences in technologies, architecture, hardware and software being used. Models for visualization need to observe key principles like: i) proper drawing, ii) shape and layout harmonization, iii) proper orientation and iv) symmetry. Aesthetically the model should be pleasant to visualize without there being overlapping nodes and edges.

There are several ways how to create a perfect model [1],[2] but what is good for one problem might not be suited for others. Some basic ideas for creating a good model are: i) perception of nodes, edges and labeling, ii) plausible diagram as regards to structure, iii) recognition of familiar structures, iv) compositionality/ability to add more building blocks as one requires, v) layout that is easy to recognize, and vi) reduced model layout.

Even though the UML and other formalisms definitely provide for representing system behavior and interaction, however at least two separate notations have to be used to represent processes and classes.

Ideally a notation that can combine both should be useful for proper process representation [1]. In a sense UML use case diagrams do this in a rather crude manner. On the other hand, Petri nets could be used, but classes would not be suitable for representing this. A possible solution to these issues is to use a Processor Net model which could also be called an Actor model. This type of model handles both the static and dynamic aspect.

3 Process Modeling Problems and Issues

Process modeling requires proper representation using consistent and tidy ways of representation [1]. Normally something like a processor model could be suitable.

Activities or tasks denote sequences of one or more actions also called operations. Particular entities are responsible for the activity, these are also called 'actors' or 'agents'. Actors can include a whole spectrum of entities ranging from physical entities like customers to complete systems that interact with another system. The granularity of process modeling is relative to what is required. The way of representing processes depends on the temporal ordering of the tasks and the particular modeling language or notation in use. When multiple actors are involved in processes, the role of the individual actors needs to be clearly specified. Today process or activity design is being given significant importance. This is evident from the work of MDE (model driven engineering) and SOA (service oriented architectures) that are process oriented.

The UML, other notations and formal languages are used to specify process models, however these have not been properly designed to focus specifically on process modeling. The result is that a disjoint or fragmented set of models are produced and many new notations and ideas are used just to represent processes. E.g. UML class diagrams, use case diagrams are more focused on static representation whilst activity diagrams and sequence diagrams focus on specific activities rather than seeing a complete process. Ideally, process representation needs convenient and systematic representation. Representation has to be simple and readable using basic identifiers for the system, human agents, machines, external systems, external agents, etc. Connectivity between the entities/ system and the process need to be clearly identified and specified. Ideally the notation used should also be formally verifiable. A mixture of textual and graphical representation is suggested.

Using the UML, FMC, design patterns, etc., some confusion arises as to which particular notation should be used [7]-[10]. Normally at least two notations are required. This would be the case for design patterns where dynamic and static representation is needed. E.g. if UML is used, class diagrams/component diagrams and activity diagrams would have to be used. Normally these notations do not focus specifically on the process modeling. A convenient way of solving these issues is normally to add proper labeling and include other structures in the notations. Unfortunately, the result is that these notations become overtly complex. If design patterns are used the representation is static. Usually pattern information gets confusingly mixed up with class information in pattern design process modeling. Unfortunately, class representation is not sufficient for explaining and exploring operational information. Notations used for pattern representation cannot associate information with real nodes like those identified in activity or actor modeling. Adding all these details is possible at the expense of creating unreadable diagrams. Many solutions do not attach information about operations and processes to a singular notation. This implies two sets of notations are required.

4 Process Modeling Solution

4.1 Patterns

Patterns in software engineering and computing are useful for solving recurring problems. Communication between different stakeholders and understanding new paradigms are possible [2]. One of the deepest problems that exist is that of representing reality using graphical notations or natural languages. System can be segmented and unified from different aspects and degrees. Patterns that are exhibited in one instance normally represent a particular temporal state which does not necessarily exist in a future configuration. Aesthetically driven design, visualization related to perception are opposed to chaotic setups that might result.

Normally in traditional approaches the focus is on rigid patterns. Repeatable patterns can be identified even in computer networks and grid computing.

Patterns represent an associative way for the evolution and storage of knowledge in the following aspects:

i) Uniformity for system comprehension
ii) Uniformity for system representation
iii) Provide for different configurations
iv) Provide for a high level of conceptual experimentation
v) Provide for Process/Actor control and modification
vi) Provide for Design and Implementation independence
vii) Extraction of abstract solutions

From a wider perspective, especially if the traditional 'system' concept is extended to include users, actors, players or external entities that act upon the system, the system configuration is dynamic and exists in relation to time. Very few symbolic notations properly explain the spatio-temporal relationships of system processes. This happens when the high level system architectural description is dependent on the type of service or job being performed by the system. i.e. more dependent on the 'actor\ model or user.

Many persons/engineers will definitely agree on the impact of effective design of software and hardware requiring proper handling at the initial stages. Modern systems are increasingly complex. Special clarity is required to understand and communicate the details prior to the design stage. Although methods like UML (Unified Modeling Language), UML-RT, DARTS (Design Approach for Real-Time Systems), HOOD (Hierarchical Object Oriented Design), JSM (Jackson Structured Method), MASCOT (Modular Approach for Software Construction Operation and Testing), etc. all help represent the system, many times better representation is needed for explaining the system at conceptual levels. Formal models have been used but most are non visual. This work attempts to show how a Processor Net model also known as the Actor model can be used to model conceptual system processes.

The Processor Net model is derived from high level Petri nets, it is also known as the Actor Model [3],[4].

5 Processor Net Patterns

It is possible in the real world to model an actor using a token or a processor or a system. But here it is preferred to use either a system or a processor because these can be easily decomposed further and offer better visibility.

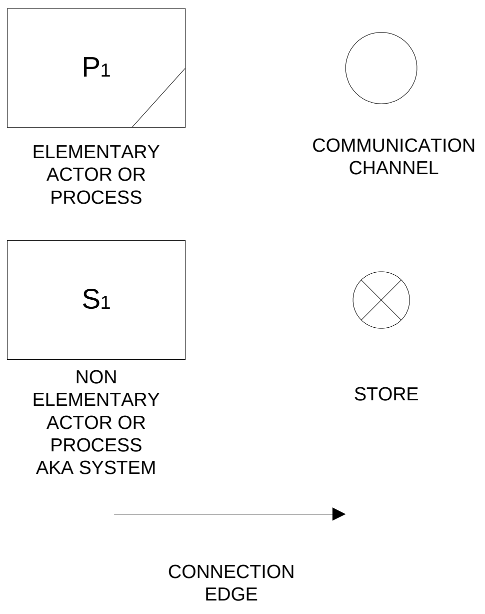
P1
ELEMENTARY ACTOR OR PROCESS
COMMUNICATION CHANNEL
S1
NON ELEMENTARY ACTOR OR PROCESS AKA SYSTEM
STORE
CONNECTION EDGE

Fig. 1 Actor Model / Processor Net Symbols

5.1 Processor Net Brief Description

The processor-net models used here are based on the actor model presented in [3].Two types of block entities are defined. i) Elementary actor or processor and ii) non elementary actor here called a system. I.e. a system is composed of a set of elementary actors or processors that are not necessary to define or represent at a high level. Then there are places. Places are represented as circles. Places also known as channels, are used to store output or input items that are produced or used by the processors as well as the system. A store is a special place type. A store is denoted as circle with an X inside. Places or stores can contain tokens in a similar concept to those in Petri nets.

The directed edges that connect the places, processors and system represent the flow of the system. The places and processors must exhibit classic high level Petri net behavior. The system entities might contain entire subnets. So their internal behavior is not as yet defined. I.e. it is abstracted. Fig. 1 shows the main symbols used for constructing a processor net model.

5.2 Task Oriented Pattern

This is the simplest and most elementary form of processing that can be identified. Here a single processor or resource is used to process a single task. The idea is extremely simple and normally very useful for small systems. At this level we do not need to show a system but just the processing because this is the actual system itself. Typical of this behavior is a simple queue serving a person or entity. A vending machine or a ticket vending machine, a simple web service, etc. are other possible examples of this.

If multiple resources are required for a task, more input nodes and edges have to be added to the processor. Fig. 2 shows task oriented behavior. The first model shows a single input, the other one shows one with two inputs.

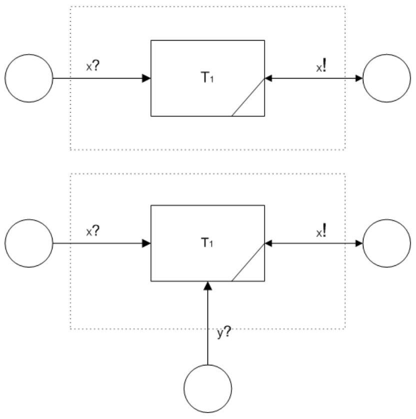

Fig. 2 Task Oriented Behavior

5.3 Producer Consumer Pattern

This type of behavior is similar to the publisher/subscriber pattern or the common producer/consumer problem. In this case, control and synchronization concepts are not included. Communication is asynchronous. Process P1 can be considered to be the main actor initiating the process. The consumer can also be considered to be an actor. Such an approach is used in event driven systems. E.g. Client/Server, Bank ATM, delivery of E-mails to subscribers or broadcasting SMS messages to mobile phone users. The model can be enhanced to get notification or reply back, but this is not always necessary. If this modification is done the behavior becomes synchronous behavior. Here there are only two actors. These can be changed. Fig. 3 shows producer consumer behavior.

5.4 Product Centered Pattern

This type of behavior represents different products or entities that have to be mapped to different processors which possibly represent objects or systems. Here a given system S1 outputs a set of different products that are treated by a specific processor from P1..Pn. The outputs are forwarded to another system S2.

The entities are mapped onto different processors because they require completely different treatment from one another initially. A similar analogy to this could be messages from a message pool have to reach different clients or servers for processing and then they can be returned to a similar pool. An industrial example of this would be the collaboration of machines for completing a set of different tasks. Another real world example is a customer order that has to be processed by several departments. E.g. accounts dept., sales dept., logistics dept. etc. Other possibilities of this layout are computer architectures, system architectures, etc. Fig. 4 shows producer consumer behavior.

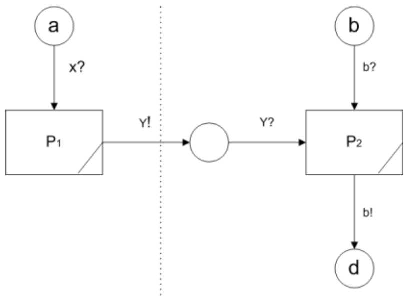

Fig. 3 Producer/Consumer Behavior

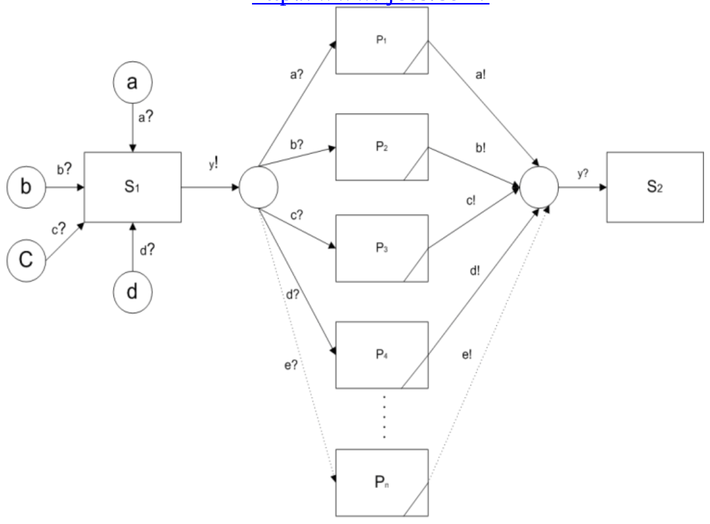

Fig. 4 Product Centered Behavior

5.5 Resources Centered Pattern

This type of behavior is the opposite of the product-centered process pattern. Here different products or entities are mapped onto the same processor or similar identical processors. There is no product or object differentiation as in the case of the product-centered approach. The processor can deal with any product or entity that arrives. It is possible to assume that products are similar in certain cases. The entities for processing are assumed to arrive from systems S1..Sn and after being processed, go to Sout1 to Soutn. P1 is the processing resource. If P1 is replaced by a system element instead it is possible to have multiple processors inside, hence multiprocessing. A classic example of this behavior is an office that services all types of requests. A processing system that handles all different orders is an alternative. Many types of information systems, computer hardware and even computer networks may have components that exhibit this behavior. Fig. 5 shows resource centered behavior.

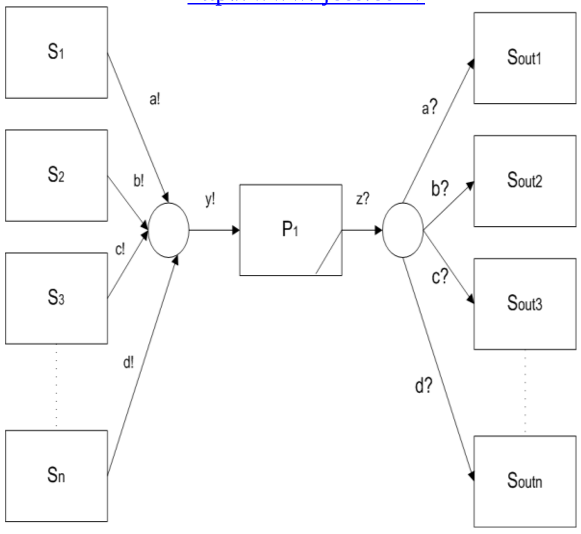

Fig. 5 Resources Centered Behavior

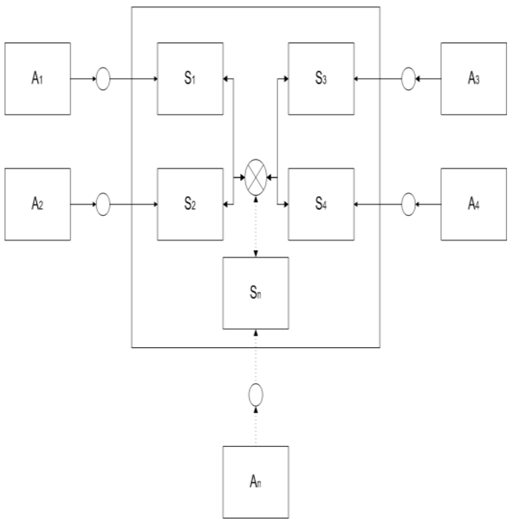

Fig. 6 Multi-Agent Oriented Behavior

5.6 Multi Agent Oriented Pattern

Different persons or agents sharing similar resources and carrying out different tasks simultaneously are typical of this type of process. Agents connect to different processors or systems that are possibly similar. The idea is based on virtualization.

There are virtual agents or resources and a virtual configuration is possible in a temporal relationship.

Agents are represented as A1..An and the system elements they connect to are represented as S1..Sn. An agent A1 connects to system S1, A2 to S2, etc. this ordering is maintained for this pattern. The virtual systems or processing elements connect to a common information or undefined data store for common information exchange. Possible examples are front end banking applications based on virtualization, internet agent technologies, parallel processing, etc. Obviously the pattern could be modified to show different types of system configuration.

6 Practical Application of the Patterns

The patterns explained previously can be used for behavioral visualization of different types of systems. Reverse engineering of existing systems or forward engineering is a useful source of information in this respect.

The models can be used for formal specification. I.e. it is possible to specify schemas for the processors and the system elements using VDM (Vienna Development Method). Finally, schemas can be produced for the entire pattern.

If more than two patterns are used to model the same system, it is possible to compare the complexity of the pattern using this simple formula.
Pattern Complexity = number of nodes + number of edges. or Pattern Complexity = number of nodes + number of edges + tokens. This type of modeling opens up the possibility for exploring new relationships from a processing perspective, instead of a more classical approach of starting off from the class diagram.

7 Conclusion

Applying process patterns prior to the design phase may result in a totally different end product. Patterns have already been successfully applied to the design phase.

The approach of applying process modeling to systems can be called a process oriented approach. This is suitable for an architectural based design approach because where there are system complexity and control requirements, there should not be separation of computation and coordination. Process centered development will mean that more emphasis is put onto understanding the main process activities and behavior. Many other traditional approaches usually start off from the system static components. Modeling of the dynamic behavior is left to the end. Modeling processes is more of a challenge than modeling static views because behavior is difficult to understand properly. On the other hand the use of process patterns would enable easier success, if a proper matching pattern is found. Some of the process patterns described do not place any requirements on the temporal ordering of events, because an event can take place at any time especially in more complex structures. Some models could exhibit cyclical behavior, whilst others are more acyclical. The system component described may contain various subset information. If this information is not available initially, the system component can still be used in an undefined manner. So the system component is like a black box with different edges that connect it to other entities.

There is the ability to collect required information from these notations. The process patterns can be used for constraint identification and understanding the interaction between different entities.

As this model uses high level Petri net theory it is not just a static representation. An actual Petri net can be obtained and used for optimization, simulation and verification. The layout and configuration could be used for finding optimal paths or sub paths.

Obviously it is possible to discover many other patterns than those described in this limited work. There could also be occasions where it could be impossible to use a particular pattern or set of patterns. In this case using the actor model can still prove useful for presenting a compact model of the main processes. This work has not included the formal aspects of the actor model which are available elsewhere.

References

1. Dong, J.,Yang, S., Zhang,K.: Visualizing Design Patterns in their Applications and Compositions, In: IEEE Transactions on Software Engineering, Vol. 33, No. 7, ISSN 0098-5589, pp. 433--453(2007)
2. Doppke, J.C.,Heimbigner, D., Wolf, A. L.: Software Process Modeling and Execution within Virtual Environments,In: ACM Transactions on Software Engineering and Methodology, Vol. 7, No. 1, pp. 1--39 (1998)
3. Van Hee, K.M.: Information Systems Engineering A Formal Approach, University Press Cambridge, UK (1994)
4. Van Hee, K.M.: Information Systems Architecture A Practical and Mathematical Approach, Technische Universiteit Eindhoven (2005), http://wwwis.win.tue.nl/~wsinhee/sm1/
5. Kristensen, L.M., Christensen, S., Jensen, K.: The Practioner's Guide to Coloured Petri Nets, In: International Journal On Software Tools for Tech. Transfer (STTT), Vol. 2, Springer-Verlag, pp. 98--132 (1998)
6. Kristensen, L.M., Jorgensen, J.B., Jensen, K.: Application of Coloured Petri Nets in System Development, In: Lecture Notes in Computer Science, Vol. 3098, Springer-Verlag, pp. 626--685, Springer, Heidelberg (2004)
7. Tabeling, P., Gröne,B.: Integrative Architectural Elicitation for Large Scale Computer Based Systems,/In: Proc. of the 12th IEEE International Symposium and Workshop on Engineering of Computer Based Systems, ECBS, pp. 51--61 (2005)
8. Gröne,B.: Conceptual Patterns,In: Proc. of 13th IEEE International Symposium and Workshop on Engineering of Computer Based Systems, ECBS, , pp. 241--246 (2006), Potsdam, Germany
9. Knöpfel, A. Gröne, B. ,Tabeling, P.: Fundamental Modeling Concepts, Wiley, pp. 1--321, West Sussex UK (2005)
10. Spiteri Staines, A.: Modeling UML Software Design Patterns Using Fundamental Modeling Concepts (FMC),In: Proceedings of the 2nd WSEAS European Computing Conference, , pp. 192--197,Malta (2008)