2011 International Conference on Computer and Software Modeling
IPCSIT vol.14 (2011) © (2011) IACSIT Press, Singapore

Formal Description of Architectural Patterns

Reshu Kapoor and Dharmendra Kumar Yadav

[1] Motilal Nehru National Institute of Technology, Allahabad, India

Abstract. Formal description of architectural patterns helps designers to detect bugs early in the design phase of Software Development Life Cycle (SDLC). The emphasis of the work is on capturing the architectural pattern and providing it's formal description in terms of concurrent Calculus Of Communicating Systems (CCS) processes. The paper presents model of architectural patterns which describes their structure as well as behavioral properties. The specification of the patterns is given in terms of Modal μ-calculus formula. The models have been implemented through parallel composition of CCS processes. This implementation of models is checked against Modal μ-calculus properties for their correctness. The patterns which have been discussed in this paper are Broker pattern and Pipes and Filters pattern. The models are verified using concurrency work bench.

Keywords: Modeling, Architectural Pattern, CCS, μ- calculus.

1. Introduction

Due to the fast growth and policy changes in the businesses, software system which models them are required to change frequently. To incorporate changes in requirements, the software developer makes changes in the software. These changes may not be local, and truly affect the software adversely. They may introduce bugs in the software. One possible solution is to localize the changes in the software; developer should be aware of these changes and incorporate them properly. The other solution is to provide architectural pattern as it captures problem at higher level of abstraction. The architectural patterns describe the problem that occur again and again and then provides a solution of the problem in such a way that one can use the solution many times, without doing it in the same way [3]. The paper is an attempt to formally capture these architectural patterns. The architectural pattern specification has been given through property in model μ-calculus [9], [12] and implementation has been done in CCS [5]. The patterns are verified using concurrency work bench [4]. The syntax of CCS is given in table1.

2. Related Work and Background

Various approaches for specifying and verifying software architectures can be found in the literature. Architectural Description Languages (ADL) based approaches consist of languages defined to describe, model and implement software architectures. A classification of ADL based languages can be found in [2][8]. Some of these languages have formal semantics supporting formal analysis. An example of this approach, Darwin[1] which uses π -calculus to model the component interaction and composition properties. Wright [11], which represents interface points as ports uses a subset of CSP for its formal semantics. It allows architects to specify temporal communication protocols and check properties such as deadlock freedom. Besides special purpose ADLs, general purpose modelling techniques such as UML have also been found to be useful for modelling high level software architectures[6], [7].

[1] Corresponding author. Tel.: + (919695870636).
E-mail address: (reshu.kapoor@gmail.com, dky@mnnit.ac.in).

CCS has been used as formal languages for defining the architectural patterns as composition of CCS processes. BASIC CCS operators have been described in table 1 where P1and P2 are two processes and 'a' and 'b' are input actions. The out put action is denoted by 'a.

Table1: Syntax and description of CCS operators

Syntax of CCS	**Description of CCS operators**
a.p1	After performing action a, it becomes process p1. '.' Operator is known as prefix operator and is used for sequencing of the operations.
p1+p2	Either Process p1 or process p2. '+' operator is used for non deterministic choice.
p1 | p2	Process p1 and p2 both execute concurrently. ' | ' operator is known as parallel composition
p1[b/a]	The action 'a' of process p1 is renamed as 'b' . '/' is known as renaming operator.
P1\a	Process p1 without an action a. '\' is known as restriction operator.

3. Broker Pattern

In the Broker pattern[10] the server registers their services at broker side and makes their services available to the client through method interface .Now when the client want to access the services it sends the request message via broker, the broker forwards the request to the appropriate server. While the client side proxy and server side proxy is used for maintaining the transparency, marshaling and unmarshaling the argument on both the client as well as the server side. The model of this Broker Pattern is given bellow an specification of the model are CCS based and properties are written in Mu calculus.

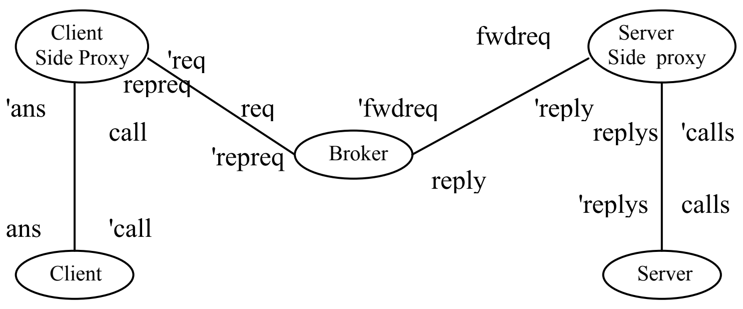

Fig. 1: Broker Pattern

3.1. Structural and Behavioral Specification

The structure and its behaviour is captured through CCS processes with the help actions performed by various component of the broker pattern. These behaviours are modelled as below.

Client ='call.ans. Client

ClientSideProxy = call.'ans.ClientSideProxy +call.'req. repreq.'ans.ClientSideProxy

Broker = req.'repreq.BROKER + req.'fwdreq. reply.'repreq. Broker

ServerSideProxy = fwdreq.'calls.replys. 'reply. ServerSideProxy + fwdreq.'calls.ServerSideProxy

Server = calls.'replys.Server

ARCH = Client | ClientSideProxy | Broker | ServerSideProxy | Server

3.2. Verification Properties:

```
prop p1 = ((not<'call>tt) \/ EF(<ans>tt))
prop p2 = A(not(<ans>tt) U <'call>tt)
prop p3 = ((not[t]<'call>tt) \/ EF((not (<'ans>tt) /\ <'req>tt )\/ (<'ans>tt /\ not (<'req>tt))))
prop p4 = maxX = <call><'req> <repreq><'ans> X
prop p5= max Y = <'call><ans>Y
prop p6 = ((not<'replys>tt) \/ EF(<calls>tt))
prop p7 = A(not(<'reply>tt) U <fwdreq>tt))
prop p8 = ((not<'fwdreq>tt) \/ EF((not (<'calls>tt) /\ <'reply>tt )\/ (<'calls>tt /\ not (<'reply>tt))))
prop p9 = max Z = <'calls><replys>Z
prop p10 = ((not[t]<'replys>tt) \/ EF((not (<'reply>tt) /\ <'call>tt )\/ (<'reply>tt /\ not (<'call>tt))))
```

The properties tabulated over can be read as:

- Whenever a client makes a call, eventually there will be an answer to the client (P1).
- Whenever there is an answer to the client, there is a prior request from the client (P2).
- Immediately after a call, either an answer or a further request will be generated from the client side proxy component (P3).
- There is a possibility of a call from client, followed by call generated from the client side proxy, followed by a reply from the server via broker and finally an answer from the client side proxy. This sequence of actions may repeat infinitely (P4).
- There is a possibility of a call from client, followed by an answer from the client side proxy. This sequence of actions may repeat infinitely (P5).
- Whenever a server side proxy makes a call, eventually there will be an reply from the server (P6).
- Whenever there is an reply from the server side proxy, there would have been prior forward the broker (P7)
- Immediately after a request from Broker proxy server either replies or further request will be generated to the server component (P8).
- There is a possibility of a call from server side proxy, followed by reply generated from the server and this sequence of actions may repeat infinitely (P9) .
- Immediately after a call, either an answer or a further request will be generated from the broker component (P10).

4. Pipes and Filters

The *Pipes and Filters* architectural pattern[10] provides a structure for systems that process a long stream of data. Each processing step is encapsulated in a filter component. Filters receive the message from the input pipe then process the message and publish the result to the output pipe. Pipes connect one filter to the next one and sends output message from one filter to next one. The model of the Pipe and Filter Pattern is given below.

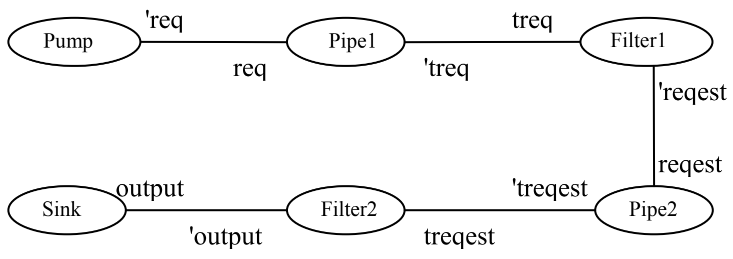

Fig. 2: Pipes and Filters Pattern

4.1. Structural and Behavioral Specification.

The structure and behaviour of the pipes and filters pattern can be illustrated through above examples. In this example we have Pump which wants to send the processed data to the Sink but the processing can be divided into sequences of operations which can be done in stages. These stages and its behaviour has been defined with the help of CCS expressions as below.

Pump = 'req.Pump

Pipe1 = req. 'treq.Pipe1

Filter1 = treq.'reqest. Filter1

Pipe2 = reqest .'treqest.Pipe2

Filter2 = treqest.'output.Filter2

Sink = output.Sink

ARCH = Pump | Pipe1| Filter1 | Pipe2 | Filter2 |Sink

4.2. Verification Properties

```
prop p1 = ((not<'treq>tt) \/  EF(<output>tt))
prop p2 =  ((not<'req>tt) \/ EF(<request>tt))
prop p3 = A(not (t<output>tt)  U (<'req>tt))
prop p4 = <'req ><'treq>><'reqest> <'treqest><'output>tt
```

The properties can be read as

- If there is a request for transferring the message, eventually there will be output message to Sink (P1).
- There is a request from Pump it will eventually reaches Pipe2 (p2).
- There will not be any output until request has been made from pump (p3).
- Sequence off streams will pass through every pipe and filter (p4).

5. Conclusion

The broker as well as pipes and filters patterns have been formally described using CCS. Mu-calculus has been used to specify these patterns. These patterns have been model checked against mu-calculus properties for their correctness. It has been found that properties mentioned are satisfied over those models. Concurrency workbench is used for formal verification of the system. These results show that specification has been met by the implementation for the patterns considered..

6. References

[1] J. Magee and J. Kramer. Dynamic structure in software architectures.*SIGSOFT Softw. Eng. Notes*. 21(6):3–14, 1996.

[2] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans.Softw. Eng.,* 26(1):70–93, 2000.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[4] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench.in Proceedings of the international workshop on Automatic verification methods for finite state systems, pages 24–37, New York, NY, USA, 1990. Springer-Verlag. New York, Inc. .

[5] R. Milner. *Communication and Concurrency*.Prentice-Hall,1989.

[6] M. M. Kand´e and A. Strohmeier. Towards a UML profile for software architecture descriptions. In A. Evans, S. Kent, and B. Selic, editors, UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings, volume 1939 of LNCS, pages 513–527. Springer, 2000.

[7] C.Lange, M. Chaudron, and J. Muskens. In practice: Uml software architecture and design description. *IEEE Software.* 23(2):40–46, March- April 2006.

[8] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*. 17(4):40–52, 1992.

[9] C. Stirling. An introduction to modal and temporal logics for CCS. In Proceedings of the UK/Japan workshop on Concurrency: theory, language, and architecture, pages 2–20, New York, NY, USA, 1991.Springer-Verlag New York, Inc.

[10] F.Bushman, R. Meunier, H. Rohnert, P.Sommeriad, M.Stall, P. Somerland. *Pattern Oriented Software Architecture*. Volume 1: A System of Patterns. John Wily & Sons, August 1996.

[11] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodology.* 6(3):213–249, 1997.

[12] D. Kozen Result on the propositional mu-calculus. Theoretical computer sci,27:33-354,1983 .