Improved Self Fused Check pointing Replication for Handling Multiple Faults in Cloud Computing

Sanjay Bansal[1], Sanjeev Sharma[2] and Ishita Trivedi[3], Mrinalika Ghosh[4]

[1] Acropolis Institute of Technology & Research
Indore, India
Sanju2.bansal@gmail.com
[2, 3] Rajiv Gandhi Prodhyogiki Vishwavidya
Bhopal, India

Abstract—The performance of checkpointing replication fault tolerance technique is severely bottlenecks due to handling of number of replicas generated for a large number of nodes to tolerate multiple faults such as multiple failure of nodes, processes etc. In fusion based approach, these checkpointing replicas stored at large number of computing nodes is aggregated into some data structure to handle efficiently through fused data structure. These impose higher overheads of fusing a large numbers of checkpointing replicas. In this paper, a self fused checkpointing replication (SFCR) for cloud computing is proposed. All checkpointing replicas assigned to store at a particular node are stored in a self-fused-shared-checkpointing-replicas file already created and located at every node rather than storing as a separate checkpointing element and than fusing. Thus, it eliminates the need of further fusing of the checkpointing replicas stored at different checkpointing replicas storage nodes, as checkpointing replicas assign to store a particular node stored in an already created fuse file at every checkpointing replicas storage node. It improves the performance without affecting the specified fault tolerant capabilities as failure of any node will result in loss of all replicas irrespective of separate checkpointing file or shared checkpointing fused file. Costs to maintain these set of self fused shared files are obviously less than the number of separate replicated files in terms of time and efforts that it takes to create, and update a file. Thus, proposed approach is enhancement of performance without compromising the specified fault tolerance capability. At the same time when system seems to be prone to many number of faults, some specific self fused shared files consist of important and critical data that can be further replicated to enhance the fault tolerant capability of a group of important and critical nodes or processes at run time. Thus, it also provides adaptive dynamic fault tolerance capability in a simple manner. Effectiveness of proposed technique is also presented

Keywords- Cloud System; Replication;, Check Pointing;, Replicated -Checkpointing.

I. INTRODUCTION *(HEADING 1)*

Cloud computing is emerging as a next generation of computing. Cloud computing, also known as a utility computing, is the highest degree of virtualization of resources like computing devices, storages, communication infrastructure etc. Cloud computing enables its user to focus on his core function rather than other related issues such as installation, configuration, investment, management etc [BP, 09]. From simple and small to large and complex, applications are shifting from traditional computing to cloud computing due to elastic scalability and performance. From letter typing in word to aerospace and scientific applications are looking for a paradigm shift to cloud computing. Due to its large scalability, fine grained billing as well as ease to use cloud computing has become the key to all computing applications. Fault tolerance in cloud computing is a major issue and checkpointing replication can be used as multiple fault tolerance technique to tolerate failure of multiple computing nodes, processes or both.

Fault tolerance is a crucial requirement in cloud computing. As the size and complexity of cloud system is increasing to provide services to millions of users with large data transfer to and from, probability of faults have also increased. Faults are now inevitable and can not be prevented totally [AA, 71]. Air traffic control, defense applications, online railway reservation system and online banking are few applications where user may be unaware of faults and may continue with his/her normal operation. Even a single fault can lead to great loss of human lives and money. In such a situation, inclusion of fault tolerance becomes essential. This inclusion of fault tolerance introduces an overhead which affects the performance of whole system. In case of multiple fault situations, it becomes more severe. In real time application, producing the output after a predetermined time for which system is designed seems to be impossible due to lack of fast and efficient recovery [SB, 2010]. The increasing frequency of occurrence of faults in increasingly-complex hardware require effective and cost-efficient fault tolerant mechanisms in today's large scale dynamic cloud systems. A cloud system must continue its normal execution even if single fault or multiple faults such as failure of multiple nodes or processes have occurred [BB, 11]. This ability of cloud system is known as multiple faults tolerance and such cloud systems are called multiple faults tolerant. Reliability and availability are two most common ways to expresses a system's ability to tolerate failure. Reliability is concerned with willingness for correct service and availability is concern with continuity of correct service [AA, 2001].

Replicated checkpointing is emerging as a scalable fault tolerance technique. [JPW, 09]. Replication is done on different computing nodes instead of dedicated check pointing server in order to reduce the checkpointing overheads. Number of replicas decides the number of faults it can tolerate. In order to tolerate f crash faults of n computing nodes n*f checkpointing replicas are required. Number of checkpointing replicas increases for number of computing nodes n and with increased fault tolerance capability. These huge numbers of checkpointing replicas require large space as well as consistency issues. The performance becomes a severe problem for large number of processes or data needed to replicate at run time again and again. Such increase of fault tolerance capability bottlenecks performance due to high overheads of maintaining larger number of replicas. Fusion Based technique is used in which all replicas which are stored at a particular node are fused into some data structure [VK, 10].

Storing checkpoint as a separate checkpoint replica and then fused introduces a high overhead and it degrades the performance severely. In this work, a self fused checkpointing replication approach is proposed which is self fused in the sense that checkpoint replicas that need to be stored at a particular node is directly stored in a common and shared fused file already created on every node, instead of storing as a separate checkpointing replica entities than fusing. Thus, high overhead is minimized and performance is improved. Since from beginning checkpoints replicas are self fused hence management of these aggregated checkpointing replicas as fused entities become easy as compared to large number of checkpointing replicas as individual entities.

II. Related work

In our work, these entities are distant checkpointing replicas. Checkpointing replication [JPW, 09] is widely used as a multiple fault tolerance technique to tolerate multiple faults like failure of multiple nodes or process. In checkpointing replication, number of replicas is very large in numbers for a large distributed computing system which consists of large number of nodes with capability to handle multiple faults. Management of these large replicas is very costly in terms of time and efforts required to store and to maintain consistency which affects the performance severely especially in faults free situations. Large number of replicas need more updates hence high consistency maintenance overhead and vice versa [HS,2010].

Fusion is emerging as an approach to handle this large number of replicas [SB.10]. Fusion is a logical combination of two or more entities [PF, 04] [PA, 05] [LN 09]. In fusion based checkpointing replication approach, large numbers of replicas is aggregated in the data structure based on some logical grouping. R replicas stored at every node are fused in some data structure and the replica located at same node is a criterion for logical grouping for fusing. P. Palacharla et al. [PP.94] proposed and discussed data fusion using fuzzy-valued logic in which data received from various sources are combined based on its quality. However, it is only suitable for small number of replicas. Derek Coleman [DC, 94] proposed fusion in object oriented context. Kumar, R et al. [kr, 03] proposed DFuse which manages distributed fusion of data streams, performing dynamic assignment of fusion operators to nodes and adapting it at run-time. R. Meier et al. proposed data fusion as a location mechanism [rm, 03]. Firat Kart et al. [fk, 07] proposed a database fusion infrastructure that produces a local database that comprises aggregated information from multiple remote databases, selected and transformed into a common table format and data representation. Maria Chtepen et al. [mc, 09] proposed adaptive check pointing and replication. Haiying Shen [HS, 10] proposed integrated file sharing approach to handle large number of replicas. I. Goiri et al. [IG, 10] proposed smart checkpoint infrastructure for virtualized service providers. It uses another union file system to differentiate read-only from read-write parts in the virtual machine image. In this way, read-only parts can be check pointed only once, while the rest of checkpoints must only save the modifications in read-write parts, thus reducing the time needed to make a checkpoint. B. Balasubramanian et al. [VG, 10] proposed fusion based techniques to deal with large number of replicas. It is emerging as a popular technique to handle multiple faults. Basically it is an alternate idea for fault tolerance that requires fewer backup machines than replication based approaches. In fusion based fault tolerance techniques, back up machines are used which is cross product of original computing machines. These backup machines are called as fusions corresponding to the given set of machines. Overhead in fusion based techniques is very high during recovery from faults. Hence, this technique is acceptable if probability of fault is low [BB, 11].

In this paper, we propose a self fuse based replicated checkpointing fault tolerance algorithm in which placement of all replicas is suggested by John Paul Walter [JPW, 09]. Replicas which are stored earlier on a node as a separate check pointing storage is now stored in a shared checkpointing which is an already created file located on every node. It eliminated need of further fusing since checkpoint replicas are already stored in a common file termed as self fused file.

Computation of location for placement of replicas is a crucial aspect for achieving the high dependability. Placing the different replicas on nearby nodes may result to breakdown just due to switch failure. In proposed approach, Placements of distinct replicas on nodes are computed by the Replicated Checkpointing Location Computation Module that uses algorithm1 proposed by John Paul Walter [JPW, 09] which also generates index file consists of a table in which there is an entry of all replicas assigned and stored as a self fused file at node. These index files are termed as, replicas-on-Ith node index (ROIN). ROIN works as a locating mechanism for all the replicas of a particular data as well as useful in maintaining consistency and finding the replicas of failure nodes to trigger the recovery. Thus, recovery that was complex in earlier fusion based approaches suggested by earlier researchers is now easy and this proposed technique can be employed in a large distributed system where probability of failure is medium. Thus, in a large distributed computing environment consists of n nodes and n self fused checkpointing replicas along with n index termed as ROIN are generated in proposed approach.

Instead of storing each replica out of r number of replicas assigned to store at a node as separate storage space entity, all replicas r of specified node is aggregated in a fused data structure in one common shared file called self fused file which is located on every node. These fused files are self fused files since there is no need to aggregate the different checkpointing replicas as a fusion process further in order to optimize the number of replicas once placed and stored on that particular node. It aggregates the replicated check pointing replica in a self fused file without compromising the fault tolerance capability even at improved performance. Since handling such small number of aggregated checkpointing replica fuse files as compared to handling of large number of replicas which require less time and efforts. In a large distributed system which consist of n nodes with f-1 fault tolerance capability will require to handle n*f number of checkpointing replicas which is very tedious tasks as compared to handling n checkpointing self fused file which is aggregated into all replicas at every node. Handling of n self fused file is very simple and economical with the help of ROIN which is meta data associated with each self fused file. Besides these from beginning, replicated check pointing of specified number of nodes are written in common storage space hence it is self generating fused check pointing replication which eliminates the need of calling a fusing process once all checkpointing replicas are stored. Thus, proposed self fused checkpointing replication serves as dual purpose storage and fusion in one, which saves the time of calling separate algorithm of write and fuse functions. This is handled by Replicated Checkpointing Storage Module (SM) which uses algorithm2 and is named as self fused storage algorithm.

Further these self fused checkpointing replicas files can further be replicated at run time on different nodes to enhance the fault tolerance capabilities at run time or when system prone to failure for a specific data. Thus adaptive run time enhancement of some critical and important data is also achieved.

III. INFORMAL ANF FORMAL DESCRIPTION OF WORK

The most import functions associated with proposed self fused checkpointing replication approach are replicas location computation and storage as aggregated replicas in a fused file already created at every node. Proposed architecture consist of major modules and these modules as shown in basic and detailed architecture in figure 1 and 2.

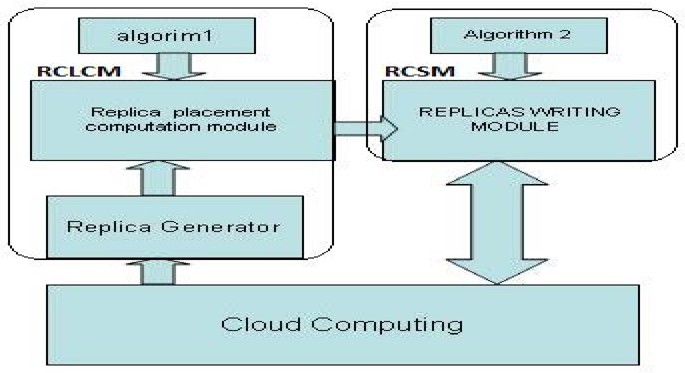

Fig1: Basic Architecture of Proposed Self Fused Replication Checkpointing

Detailed architecture with fault tolerance and replication manger is shown in figure 2

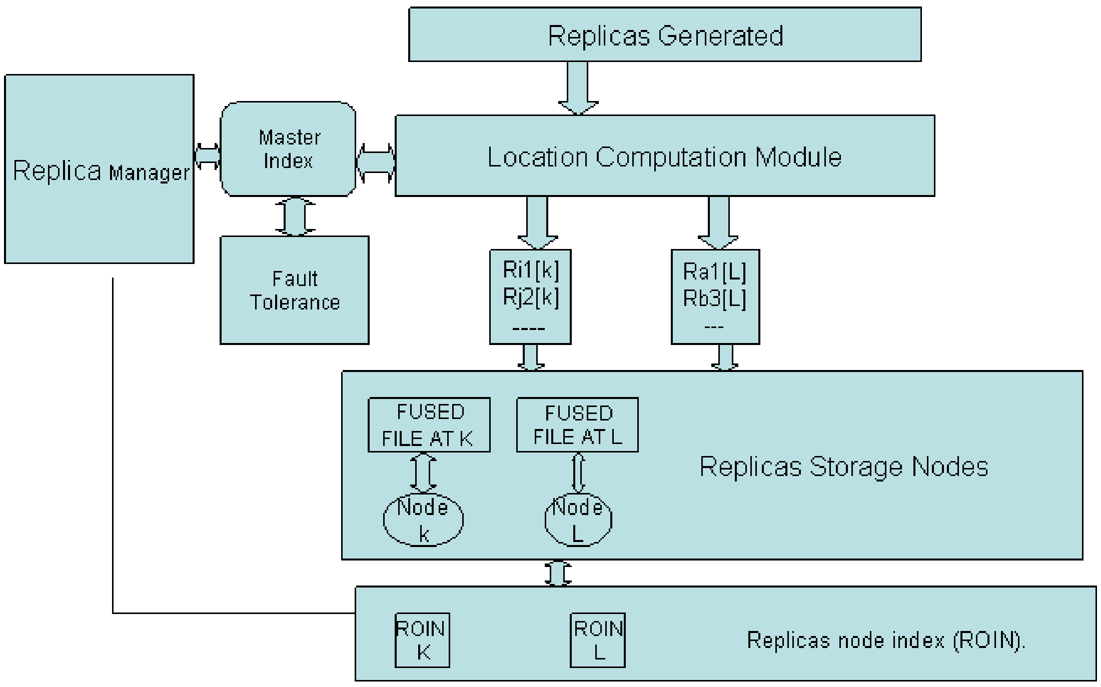

Figure 2: Detailed Architecture of Proposed Self Fused Checkpointing Replication

A. . Replicated Checkpointing Location Computation Module (RCLCM):

RCLCM computes the location of various replicas of nodes and processes generated by replica generation module. The module takes number of nodes on which checkpointing replicas can be stored as well as number of replicas per node or process in order to tolerate a specified number of faults in outputs as valid replicas that need to store for all nodes. This module is determines the location of a set of replicas of all nodes with improved constraints, suggested by John Paul Walter [JPW, 09]

1. A node should not replicate to itself.
2. A node should replicate to exactly r nodes, each of which may only store r replicas. In some cases nodes can replicate more or less nodes.
3. Each of a node's r replicas should be unique.

This module uses algorithm 1 as suggested by John Paul Walter.

```
Algorithm 1: Compute random replica placements ()
Input: Integer r, the number of replicas
Input: Integer n, the number of nodes
Output: Replica array, Valid Replicas [0...n - 1] [0...r - 1]
1: for all i such that 0 = i < n do
2: Preload node i's replicas with i
3: end for
4: Circular-shift each column (j index) of Replicas
   by column index - 1
5: for all i such that 0 = i < n do
6: for all j such that 0 = j < r do
7: repeat
8: z = random node, s.t. 0 = z < n
9: v = Replicas[z] [j]
10: until z = i
11: if v = i and Replicas[i] [j] = z then
12: valid replica = 1
13: for all k such that 0 = k < r do
14: if Replicas[i] [k] == v or Replicas[i] [j] == Replicas[z] [k] then
15: valid replica = 0
```

```
16: end if
17: end for
18: if valid replica then
19: Replicas[z] [j] = Replicas[i] [j]
20: Replicas[i] [j] = v
21: end if
22: end if
23: end for
24: end for
```

B. Replicated Checkpointing Storage Module (RCSM):

This module stores checkpointing replicas on various nodes as per the locations of placement computed by RCLCM module. Thus, the main function of this module is store all distinct replicas of a particular node in a common shared self fused replication checkpointing file already created on every node. This is done for all storage nodes. Thus, this produces a set of self fused checkpointing replication files in which each one aggregates all distinct r replicas assigned for a node. This module stores all replicas to be placed on a node in a common storage file as per computation performed by module RCLCM and uses algorithm 2 as follows:

```
Algorithm 2: self fused storage algorithm.

Input: Replica location[j] [i]

Output: Self Fused Checkpointing Fils[i], INDEX [I] =Reilcas on Ith node

1: for all i such that 0 = i < n do
2:  create cheeckpointfile[i]
3: end for
4: for all i such that 0 = i < n do
5:  valid replicas for I node= Compute random replica placements ();
6. Index [i]  =all valid replicas;
7. Master_Index=Index[i];
7. Write all replicas of  Ith node on checkpointfile[i]
8: end for
```

C. Replica Manager

Replica manager consist of various replica services and replica access services. The major services that are provided by replica manager are consistency service, core replica creation, deletion and besides these security and authentication associated with replicas are also performed by replica manager.

D. Master index

Master index module stores all meta data about location of all replicas of all nodes. This module has a table which consists of all entries of locations of replicas in a tabular format. The basic purpose of this module is to find out replica location of all replicas of an application node. Master Index is replicated to prevent single point failure.

E. Secondary Index of nodes

Secondary index is associated with every node. The basic purpose of this is to find out the information of all replicas stored on that node. Secondary Index is used to transfer the replicas on other nodes in case that node is leaving form dynamic distributed system. In case of failures it also helps to generate and locate replicas which were stored on this node.

F. Fault Tolerance

Fault tolerance modules consist of two major functions: failure detection and recovery. Failure detection module is a building block and detects the failure of nodes timely and accurately. On detection of node failures, the recovery module searches the replica locations where extra replicas of failure nodes are available by searching in master index table. After finding the location of replicas of failure nodes, it uses one of the same replicas as a primary replica and restores the normal state of execution. It also informs replica generation module to generate replicas of failure nodes, again to ensure same level of the fault tolerance of failure of nodes in future:

IV. EXPRIMENT RESULT

We have performed an experiment on 150 nodes. Cloud environment is set up by simulator which is designed using JAVA. The simulation creates an interactive cloud environment that shows the dynamic changes also. Results obtained are displayed in following Table 1.

TABLE I. PERFORMANCE COMPARISON

No. of nodes or process	**No. of faults that can be tolerated**	**Fusion Based Checkpointing replication**		**Self fused Checkpointing replication**	
		No. of files	***Time (ms)***	***No. of files***	***Time (ms)***
2	2	6	24.3	3	21.3
4	3	16	72.2	8	62.7
8	4	40	225.9	20	198.9
8	5	48	309.89	24	240
10	5	60	603.7	30	454.6

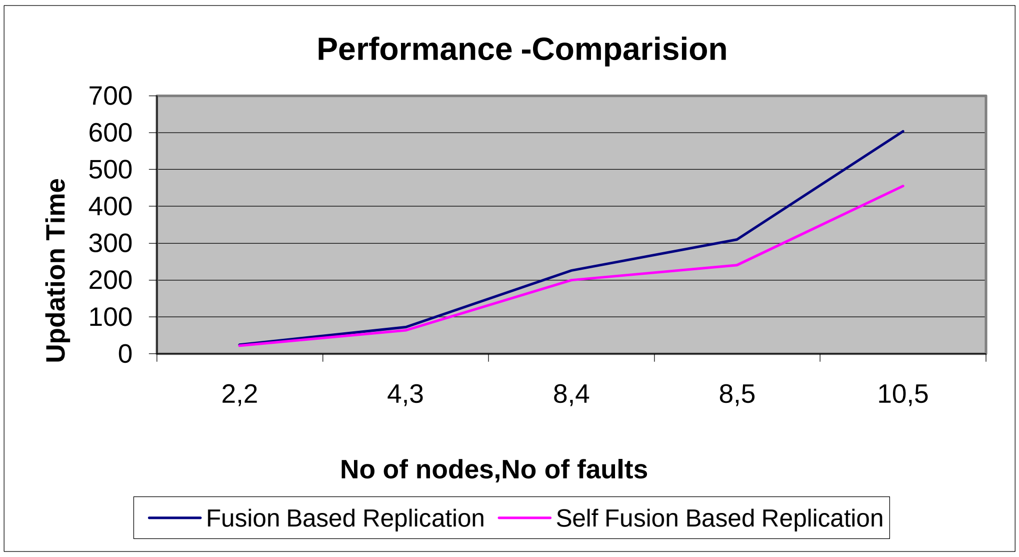

Figure 3: Performance Comparison

V. CONCLUSION

Result obtained from experimental set up shows that there is improved performance because of creating as well as writing checkpointing from beginning which requires less time as compared to creating, writing and than fusing. Storing the checkpointing of specified number of nodes as a replication in a single file from beginning reduces the overheads of creating separate files as compared to single file. At the same, time since number of replicas are same in both the case so fault tolerant capability is same but with improved performance

REFERENCES

[1] [JP, 09] J. Walters and V. Chaudhary, "Replication-Based Fault Tolerance for MPI Applications," IEEE Transactions on Parallel and Cloud Systems, Vol. 20, No. 7, July 2009.

[2] [AA, 71] A. Avizienis, "Fault-Tolerant Computing: An Overview," IEEE Computer, vol. 4, no. 1, pp. 5–8, Jan 1971.

[3] [VK, 10] V.K Garg, “Implementing fault-tolerant services using fused state machines,” Tech-nical Report ECE-PDS-2010-001, Parallel and Cloud Systems Laboratory, ECE Dept. University of Texas at Austin (2010)

[4] [BB, 11] B. Balasubramanian, V. K. Garg, “Fused Data Structures for Handling Multiple Faults in Cloud Systems,” 2011 31st International Conference on Cloud Computing Systems, 1063-6927/11 $26.00 © 2011 IEEE.

[5] [Xt, 04] X. Tang and S. T. Chanson, “Minimal Cost Replication of Dynamic Web Contents under Flat Update Delivery,” IEEE Transactions on Parallel and Cloud Systems, vol. 15, no. 5, May 2004.

[6] [IG, 10] ´I˜nigo Goiri, Ferran Juli`a, Jordi Guitart, and Jordi Torres, “Checkpoint-based Fault-tolerant Infrastructure for Virtualized Service Providers,” 978-1-4244-5367-2/10/$26.00 c_2010 IEEE.

[7] [PP, 94] Prasad Palacharla, Peter C. Nelson AND Virginia P. Sisiopiku, “Data Fusion Using Fuzzy-Valued Logic,” Intelligent Vehicles '94 Symposium, Proceedings of the , Oct. 1994, pp: 115 - 119 IEEE.

[8] [MC, 09] Maria Chtepen, Filip H.A. Claeys, Bart Dhoedt, Filip De Turck,

[9] iet Demeester, Senior Member, IEEE, and Peter A. Vanrolleghem, “Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids,” IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 20, NO. 2, FEBRUARY 2009

[10] [FK, 07] Firat Kart, L. E. Moser, P. M. Melliar-Smith, “Database Fusion Using Atom,” 1-4244-1364-8/07/$25.00 ©2007 IEEE

[11] [PF,04] Pfleger, N, "Context Based Multimodal Fusion". ICMI 04. Pennsylvannia, USA, ACM: pp. 265 - 272.

[12] [LN 09] Lalanne, D., Nigay, L., et al.. " Fusion Engines for Multimodal Input: A Survey". ACM International Conference on Multimodal Interfaces. Beijing, China: pp. 153-160.

[13] [PA, 05] Pérez, G., Amores, G.,. "Two strategies for multimodal fusion". ICMI'05 Workshop on Multimodal Interaction for the Visualisation and Exploration of Scientific Data. Trento, Italy, ACM.

[14] [BP, 09] B.P. Rimal, E. Choi and I.Lumb , “A Taxonomy and Survey of Cloud Computing Systems,” 2009 Fifth International Joint Conference on on INC, IMS and IDC, 978-0-7695-3769-6/09 $26.00 © 2009 IEEE..

[15] [KR, 03] Kumar, R., Wolenetz, M., Agarwalla, B., Shin, J., Hutto, P., Paul, A.,and Ramachandran, U., “Dfuse: A framework for distributed data fusion,” in ACM Conference on Embedded Networked Sensor Systems, 2003.

[16] [RM ,03] René Meier and Vinny Cahill, “Location-Aware Event-Based Middleware: A Paradigm for Collaborative Mobile Applications?,”

[17] [DC , 94] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes and Paul Jeremaes. Object-Oriented Development: The Fusion Method. Prentice Hall, 1994.