Self-Recovery Technology in Distributed Service-Oriented Mission Critical Systems for Fault Tolerance

Raymundo García-Gómez, Juan Sebastián Guadalupe Godínez-Borja, Pedro Josué Hernández-Torres, Carlos Pérez-Leguízamo
Central Bank of Mexico (Banco de México)
Av. 5de Mayo #2, Mexico City, 06059, MEXICO
E-mail: {rgarciag, jgodinez, pjhernan, cperez}@banxico.org.mx

***Abstract*—Mission Critical Systems (MCS) require continuous operation since a failure might cause economic or human losses. Autonomous Decentralized Service Oriented Architecture (ADSOA) is a proposal to design and develop MCS in which the system functionality is divided into service units in order to provide functional reliability and load balancing; on the other hand, it offers high availability through distributed replicas. A fault detection technology has been proposed for ADSOA. In this technology, an operational service level degradation can be detected autonomously by the service units at a point in which the continuity of the service may be compromised. However, this technology is limited because it requires human supervision for recovery. In this paper, we propose an autonomous recovering technology, which detects and instructs to service units to be gradually cloned in order to recover the operational service level. A prototype has been developed in order to verify the feasibility of this technology.**

Keywords-Service continuity; fault tolerance; service-oriented architecture; autonomous decentralized systems; fault detection; fault recovery

I. INTRODUCTION AND MOTIVATION

In the presence of a failure, most of the conventional systems implement reactive fault detection and recover mechanisms either automatically or manually. In both cases, the aim is to switch to a redundant or standby computer server upon the failure or abnormal termination of the previously active system. In some cases, the Mean Time to Recovery (MTTR) [15] of these technologies may represent a low risk for the service that the system offers. However, since a failure in MCS may provoke fatal consequences, it is important to reduce the MTTR to a value near to zero

In this paper, we briefly present ADSOA [4][5][6], which has been proposed as a service-oriented architecture for designing MCS, and it has been mainly utilized in financial sector applications. This architecture provides high functional reliability since it is possible to distribute and replicate the functionality of a system in specialized service units. One of the main technologies of ADSOA, called Loosely Coupled Delivery Transaction and Synchronization Technology [6], allows the system to detect when the provision of a service has reached a point in which the continuity of the service may be compromised and it sends a signal alarm to a monitor. This approach may represent a risk for a MCS since it depends on human intervention for taking the necessary actions to repair the system.

This has motivated this paper which presents a technology to autonomously detect and recover gradually all the unit services required for the operational service level in ADSOA. This technology is based on a cloning mechanism that is activated once the operational service level has been compromised due to some failed services units. We describe the protocol and algorithms that the healthy services units utilize in this cloning mechanism and show how they coordinate among them in order to avoid a massive creation of replicas. We developed a prototype in order to illustrate this approach.

The rest of this paper is organized as follows: In Section II, we show the related work. In Section III, we give a view of ADSOA concept and architecture. In Section IV, we present the proposed technology. In Section V, we show a prototype, and finally, in Section VI, the conclusion and future work.

II. RELATED WORK

Cloning technologies have been widely used in different technological areas for providing high reliability to the system in which it is applied. In Optical Burst Switching (OBS) Networks, burst cloning has been proposed as a proactive loss recovery mechanism that attempts to prevent burst loss by sending two copies of the same burst, if the first copy is lost, the second copy may still be able to reach the destination [9][10]. When designing cloning technologies one relevant issue that has to be considered is the resource utilization by the new clones. In this sense, in OBS Networks some technologies have been proposed for optimizing such resource utilization and maintaining a QoS [11][12]. In Multi Agents Systems (MAS), a frequently proposed solution to avoid performance bottlenecks due to insufficient resources is cloning an agent and migrate it to remote hosts [13][14].

Our approach is also comparable to the existing work on cloning technologies in terms of concept and objectives but applied to a novel service-oriented architecture for MCS. The main contribution of the proposed cloning technology are the protocol and algorithms that services units utilize to detect some failures in the service provision and the way they coordinate themselves to recover gradually the operation of that damaged part of the system.

III. Autonomous Decentralized Service oriented architecture

A. ADSOA Concept

A proposal used to implement MCS in financial sector is ADSOA [4][5][6], it provides load balancing and functionality, high availability and service-oriented modeling. ADSOA is based on Autonomous Decentralized Systems (ADS) [1][2][3] and Service Oriented Architecture (SOA)[7][8].

ADS is based in the concept of analogizing living things. The living body is built of numerous cells, each growing up independently, maintaining by body metabolism, and going on living. Depending on concept of perceiving the system that it is consisted by autonomous controllability and autonomous coordinability, ADS technology achieves fault tolerance, on-line maintenance and on-line expansion of the computer system. On the other hand, SOA offers a system modeling oriented to services and allows composition and reusability. ADSOA is the combination of SOA concept with ADS characteristics.

The ADSOA conceptual model, shown in Figure 1, is composed of autonomous entities that offers or requests services via messages. Each entity is formed by several instances fully independent. Each instance has the same functionality that its entity represents. A subsystem can be formed by a group of entities and in the same sense a business may be formed by a group of subsystems. This is similar to a living organism where an instance is like a cell, a subsystem could be an organ and the business is like a living organism.

In order to model a MCS using ADSOA, it is necessary to have a service-oriented thinking. At the beginning the system architect identifies the businesses involved in the process and then models the sub-systems in a business according to their responsibility. Finally, entities are modeled according to their atomic functionality. This modeling will allow to the system to grow, evolve, do composition and reuse the components. The next phase is to develop the services entities.

All the systems immersed in ADSOA are able to configure according to physical resources and criticality level. To offer high service availability, it is necessary to have a distributed environment and put on replicated entities. On the other hand, for load balancing it is necessary to divide the functionality in the entities, in such a way that the work be split without a coordinator. The challenge is to provide auto-coordination and auto-control to the system. In this sense, the Autonomous Processing Entity (APE) was proposed; it implements the communication protocol, manages the control instance messages and the services execution. Also, it is possible to define in each service (offered or requested) of the APE its criticality. All these elements form a technology denominated “Loosely Coupling Synchronization and Transactional Delivery Technology”.

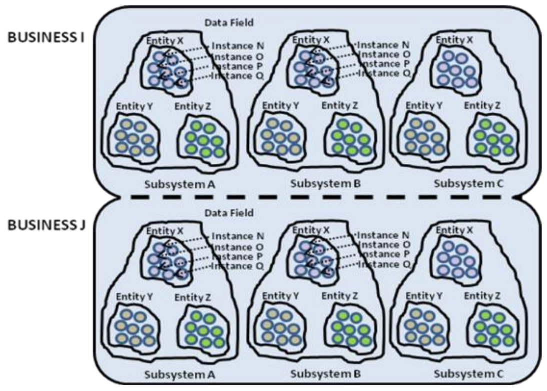


Figure 1. ADSOA Conceptual Model

B. Loosely Coupling Synchronization and Transactional Delivery Technology

In this technology, we define the concept of transaction in the scenario in which an entity requests a service to another and requires knowing if it has been received. The requesting entity must maintain this request in pending processing state until it receives an acknowledgement from receiving entity. Also, we define sequential order in the sense that the entity requester must receive a minimum number of acknowledgments from receiving entities in order to send the next service request, for example, a *X+1* request should not be sent until it receives the minimum number of acknowledgments of the *X* request.

The service request information structure should include the following elements: Content Code, Foliated Structure and Request Information.

The Content Code specifies the content and defines the requested service.

The foliated structure identifies the transaction. This structure is based on:

1. requester id,
2. specialized task id for that request (Pivot),
3. a sequence number,
4. a generated id based on the original request information (event number) and
5. a dynamic and unique id for the instance of the entity (instprintid).

With these elements the identification of acknowledgments received by the entity is guaranteed. We can also ensure the sequence of multiple requests, as shown in Figure 2.

If an instance receives a service request with a sequence number greater than expected, then by the principle of sequential order, knows that another instance of its entity will have the missing messages. In this case, the receiver instance asks to his entity the missed messages, that is, the other instances of the same entity. This idea is represented in Figure 3.

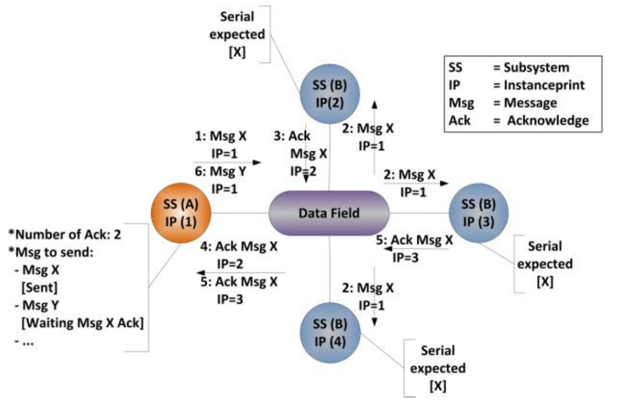


Figure 2. Sequentiality and Transactionality

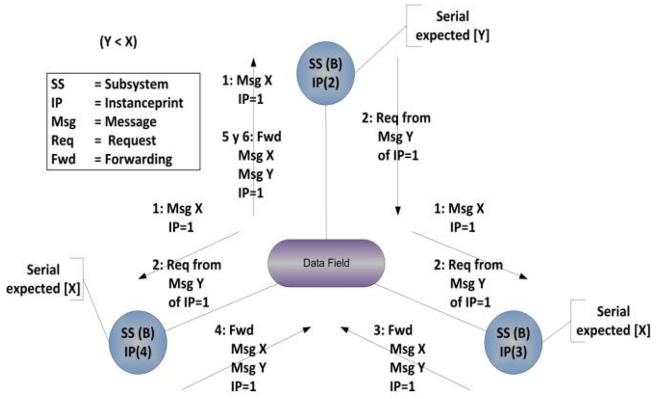


Figure 3. Synchronization with other Instances

On the other hand, if an entity receives several times the same service request, this can be distinguished by the instprintid if this request belongs to the same requester instance or from a different instance of the same entity. According to this, the receiver entity can determine whether requests received are in accordance with the minimum number of requests that the requester entity are required to send, as shown in Figure 4.

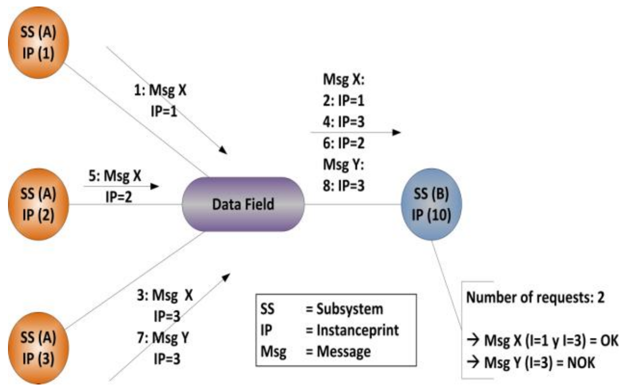


Figure 4. Receiving Multiple Requests from an Entity

In resume, the communication among the elements and its instances is based on an asynchronous and event-driven message protocol. This technology detects if an entity does not provide the service level required. It occurs when an instance sends a service request to an entity and each entity instance receives it and send back an acknowledge message, then sender registers how many acknowledges have received and evaluates if it covers the criticality level, if it is not proper, the sender repeats the sending process. E.g. consider a service with a criticality level equals to 3, its means that this business requires at least three distributed instances; when another instance requests a service to them, it expects at least three acknowledges to satisfy the criticality level, if it is not satisfied the entity will send the request of service again. When the sender detects that the maximum number of retries has reached, it triggers the alert process, which consists in sending an alert message that could be processed by a monitor element. This monitor alerts ADSOA infrastructure managers to perform the necessary activities and recover service continuity (creating new instances required to reach the criticality level). Unfortunately, this goes against MCS's principles since manual intervention is required thereby MTTR becomes dependent on operator's reaction.

In the next section, we present a technology that allows ADSOA subsystems to autonomously detect and recover for a failure in a replicated entity by cloning one by one an operational entity until the system reaches the criticality level required.

IV. Self-Recovery Technology in Distributed Service-Oriented Mission Critical Systems for Fault-Tolerance

This technology is created to allow an MSC that uses an ADSOA infrastructure to self-recover automatically. This basic operation is to use the current self-monitoring scheme and instead of sending alerts to the operator when the service level is not appropriate, it instructs one entity of the degraded group to clone itself (functionality and state). An important challenge in the cloning process is to avoid the generation of multiple indiscriminate copies, which in a living organism would be a cancer. To ensure the healthy recovery, the entity selected to recover the system, generates a cloning-key with information of the times it has been cloned, its id, its instprintid and the requested entity id; this information is introduced into the algorithm to generate the cloning-key, that will be unique to only one cloning process between this entity and the requested id.

In this architecture, all the entities offer and request services, one of this services is the recovering by cloning an entity. In self-recovering technology at least two entities are involved; to explain the protocol let's imagine a group of entities ("A subsystem"), which request a service to other group of entities ("B subsystem"). In Figure 5, "A subsystem" is requesting a service to "B subsystem", the message exchange is carried out in compliance with ADSOA Loosely Coupling Synchronization and Transaction Delivery Technology, with the number of acknowledgments needed to ensure that the level of service is appropriate for. In this example, the "A subsystem", requires 3 acknowledgments by "B subsystem", and the "B subsystem" needs 2 service requests by “A subsystem”; when the number of acknowledgments is complying, “B subsystem” attends “A subsystem”.

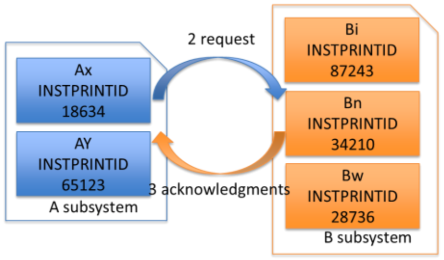


Figure 5. Normal Operation Cycle

If the number of the entities of the “B subsystem” is decreased because of a failure in the process or the server, the “A entities” detects that the service level is not complying within “B subsystem”, since the minimum number of acknowledgments, 3 for this example, cannot be reached within a specific period of time. Thus, the “A entities” starts the recovery mechanism instead of sending alerts.

Figure 6 shows the first steps in the recovery protocol. Firstly, the “A subsystem“ receives the acknowledgments from “B subsystem”. Secondly, based in the lowest “B”’s instprintid all the “A entities” select one healthy entity, which will be responsible for cloning itself. Thirdly, all the “A entities” request the "Auto-Cloning Service (reqidclon)", with the instprintid of the “B entity” selected for auto-cloning. In this example the “Bi entity” will be the responsible for cloning itself; although the “Bn entity” received the same request, only the “Bi entity” will clone. Fourthly, when the “Bi entity” receives the reqidclon request, it generates a cloning-key and sends both this cloning–key and its instprintid as a “Send the key (sendkey)” request service message. By sending its instprintid it can be ensured that the “A subsystem” will instruct to only the selected “B entity” to continue with the cloning process. Fifthly, when the “A subsystem” receives the sendkey service request, it takes the cloning-key in the message and sends it by the “Automatic recovery (autrecov)” request service message to the “B subsystem”, it also attaches to this message the instprintid selected in the second step of this protocol.

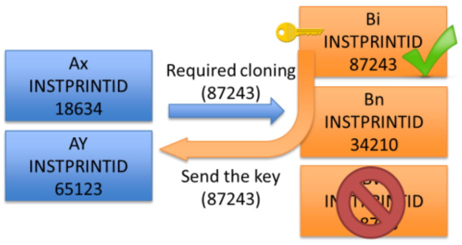


Figure 6. Start up the cloning mechanism

Figure 7 shows the final step of the protocol. When the “B subsystem” receives the autrecov request service message, as it occurs in the third step of this protocol, only the “Bi entity” will attend it, since its instprintid is in the received service message. “Bi entity” will validate if the cloning-key in the message is still valid and if so it will make a cloning of itself. During this process, “Bi entity” will close all the communication with outside and generate a new element in the same state like itself; once the cloning process is finished, it will open the communication again. Otherwise, if the cloning-key is not longer valid because the cloning process has already been completed, the message is ignored. It is important to notice, that the others “A entities” also send this final request service message, but only the first message which reaches “Bi entity” will be processed.

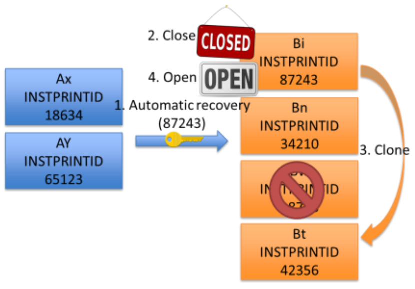
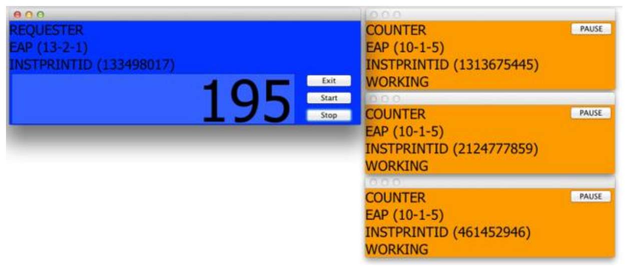


Figure 7. Cloning phase

In this sense, this technology will autonomously maintain the operational service level without human intervention.

V. PROTOTYPE

A prototype which implements this technology has been developed, as shown in Figure 8. This prototype consists of two subsystems with one entity each one, the Requester subsystem/entity, which is shown in blue color, and the Counter subsystem/entity, which is shown in orange color. The Requester demands a service to the Counter for providing a number which later it will be displayed in its screen. When the Counter receives this service request, it will increase by one the previous sent number and send it into a service request message to the Requester. For this example, the service level operation was set to 1 to the Requester and 3 to the Counter; it means that there will be only 1 instance of the Requester and 3 instances of the Counter. On the other hand, the Requester will send its next service request only if it receives from the Counter instances 3 acknowledges for the current request. In order to simulate a failure in a Counter instance, a PAUSE button has been implemented.


REQUESTER
EAP (13-2-1)
INSTPRINTID (133498017)
195

COUNTER
EAP (10-1-5)
INSTPRINTID (1313675445)
WORKING

COUNTER
EAP (10-1-5)
INSTPRINTID (2124777859)
WORKING

COUNTER
EAP (10-1-5)
INSTPRINTID (461452946)
WORKING

Figure 8. Normal Operation Cycle I (Prototype)

In Figure 9, it is shown that when the entity “2124777859” is stopped, the cloning-mechanism detects such failure and selects entity “1313675445” to repair the system. The reqidclon service request message is sent from the Requester to the Counter with instprintid “1313675445”. This entity generates the cloning-key and then it sends the sendkey service request message to the Requester.

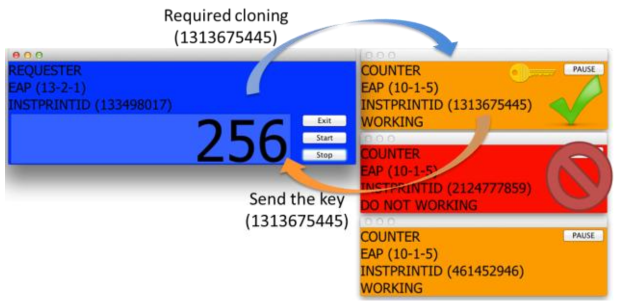


Figure 9. Start up the cloning mechanism (Prototype)

In Figure 10, the final part of the mechanism is shown. The Requester processes the sendkey service request message and sends to the selected entity the autrecov service request message. The “1313675445” entity starts the cloning process; firstly it closes the communication with outside, then it clones itself, and when the entity “191232582” is started, the “1313675445” entity finally opens the communication. In this sense, the system has repaired autonomously the damaged part and it can continue its normal operation.

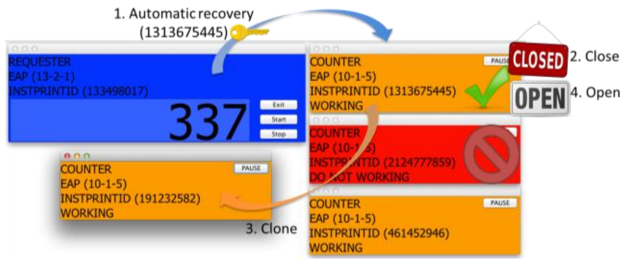


Figure 10. Normal Operation Cycle II (Prototype)

VI. CONCLUSION AND FUTURE WORK

In this paper, we briefly presented ADSOA, which has been proposed as a service-oriented architecture for designing MCS, which has been mainly utilized in financial sector applications. We have proposed a cloning mechanism to recover quickly and efficiently the operational service level when a decrease on it is detected. We have built a prototype to verify the feasibility of this technology.

Besides the ongoing development efforts to complete the cloning prototype implementation, future work in this area focuses on get some metrics about resource utilization, network partition and multiple clones’ coexistence. We will also compare the proposed technology with others such as those mentioned in Section II.

REFERENCES

[1] K. Mori, S. Miyamoto, and H. Ihara, "Proposition of Autonomous Decentralized Concept", Journal of IEEE Japan, vol. 104, no. 12, pp. 303-310, 1994.

[2] H. Ihara and K. Mori, “Autonomous Decentralized Computer Control Systems”, IEEE Computer, vol. 17, no. 8, pp. 57-66, 1984.

[3] K. Mori, “Autonomous Decentralized Computer Control Systems”, First International Symposium on Autonomous Decentralized Systems (ISADS’93), Kawasaki, Japan, pp. 28-34, 1993.

[4] L.C. Coronado-García and C. Pérez-Leguízamo, “A Mission-Critical Certification Authority Architecture for High Reliability and Response Time”, IJCCBS Special Issue on Autonomous Decentralized Systems in Web Computing, vol. 2, no. 1, pp. 6-24, 2011.

[5] L.C. Coronado-García, P.J. Hernández-Torres, and C. Pérez-Leguízamo, “An Autonomous Decentralized System Architecture using a Software-Based Secure Data Field”, The 10th International Symposium on Autonomous Decentralized Systems (ISADS’11), Kobe, Japan, 2011.

[6] L.C. Coronado-García, P.J. Hernández-Torres, and C. Pérez-Leguízamo, “An Autonomous Decentralized Service Oriented Architecture for High Reliable Service Provision”, The 10th International Symposium on Autonomous Decentralized Systems (ISADS’11), Kobe, Japan, 2011.

[7] Thomas Erl, 2005, “Service-Oriented Architecture (SOA): Concepts, Technology, and Design”, Ed. Prentice Hall.

[8] Nicolai M. Josuttis, 2007, “SOA in Practice: The Art of Distributed System Design”, Ed. O'Reilly Media.

[9] H. Xiaodong, V.M. Vokkarane, and J.P. Jue, “Burst cloning: a proactive scheme to reduce data loss in optical burst switched networks”, IEEE International Conference on Communications (ICC’05), Seoul, Korea, 2005.

[10] S. Riadi and V-A. Mohammed, “A decision algorithm for efficient hybrid burst retransmission and burst cloning scheme over star OBS networks”, Second International Conference on Innovating Computing Technology (INTECH’12), Casablanca, Morocco, 2012.

[11] L. Ji and K.L. Yeung, “Burst cloning with load balancing”, Optical Fiber Communication Conference (OFC’06), Anaheim, California, 2006.

[12] S. Askar, G. Zervas, D.K. Hunter, and D. Simeonidou, “Classified cloning for QoS provisioning in OBS networks”, The 36th European Conference and Exhibition on Optical Communication (ECOC’10), Turin, Italy, 2010.

[13] O. Shehory, K. Sycara, P. Chalasani, and S. Jha, “Agent cloning: an approach to agent mobility and resource allocation”, IEEE Communications, vol. 36, no. 7, pp. 63-67, 1998.

[14] D. Ye, M. Zhang, and D. Sutanto, “Cloning, Resource Exchange and Relation Adaptation: An Integrative Self-Organisation Mechanism in a Distributed Agent Network”, IEEE Transactions on Parallel and Distributed Systems, vol. PP, no. 99, pp. 1, 2013.

[15] P.A. Laplant and S.J. Ovaska, 2011, “Real Time Systems Design and Analysis“, Ed. Wiley-IEEE Press.