DOI: 10.15439/2017R15 ACSIS, Vol. 10 ISSN 2300-5963

Designing Graceful Degradation in Software Systems

Rohit Dhall
Enterprise Architect, Engineering and R & D services,
HCL Technologies
Noida, India
e-mail: dhall.r@hcl.com

Abstract— **Graceful degradation is an aspect of a fault tolerant software system, where in case of some failures, system functionality is reduced to a smaller set of services/functionalities that can be performed by the system. During the period of graceful degradation, system runs and offers only the minimal set of critical services, thus avoiding total outages. This paper discusses some design approaches which can be used in a software system to handle graceful degradation. It also proposes and discusses the design of the 'Capability Determination Model' (CDM), and how this model can be used to build and implement an IT software system with graceful degradation. This paper presents a high level design of the CDM and how different aspects of graceful degradation can be built into a software system using this model (CDM).Working of different components of CDM is also discussed. In the end, this paper talks about some of the alternatives of graceful degradation design and challenges associated with designing software systems with graceful degradation.**

***Keywords**- Graceful Degradation, Performance Engineering, Fault tolerannt system Design, System Dependability, Graceful Degradation Design.*

I. What is Graceful Degradation?

With software systems becoming more and more complex and performing some of the most critical operations of an organization, one thing which hasn't changed at all is the possibility of occurrence of an event, which can prevent the system from performing all its expected functionalities. This event or exception scenario can prevent the system from performing its overall responsibilities and can cause a huge loss to the businesses, which are becoming more and more dependent on the IT systems.

To minimize the risks associated with errors scenarios/outages, multiple techniques are used to make a system fault tolerance. E.g. business may decide to have a failover/HA components or to have a complete DR site at an alternate physical location, data replication and so on. But as the system becomes more complex (more subcomponents, more external integrations and so on), having a failover for complete system can turn out to be very costly in terms of money or effort, needed to build a such failover instance.

Graceful degradation refers to an aspect of fault tolerant system design, where during the occurrence of an exceptional event, system capability is gracefully lowered in terms of the services it offer or perform(in terms of number of business use cases it support or the NFRs like throughput, response time etc.).Thus a gracefully degraded system will not be providing complete set of functionality, which it generally provides in a fully functional state.

Concept of graceful degradation design is used in a wide variety of systems and its meaning can vary from system to system. For example, one system can consider graceful degradation as offering a smaller set of UI features, depending upon the capabilities offered by the browser being used by the end user, another system can consider graceful degradation as rendering a lower resolution image, a telecom system switching from 4G to 3G or 2G services, while some systems may consider graceful degradation as offering only a limited subset of critical services while making other less critical services unavailable in case of an error event. This paper considers all such cases as design aspects of graceful degradation.

Fig. 1 summarizes the concept of graceful degradation. The chart on the left side shows a system, working with 100% capabilities in a normal scenario. The chart on the right side, shows a system, which is now offering 80% of the overall capabilities, in case of an event. The system handles this degradation gracefully, by stopping the offering of certain less critical services, so that other critical services can still continue to perform. Note that this graceful degradation can also happen in the form of degradation of performance, throughput, response time and other such aspects of the system.

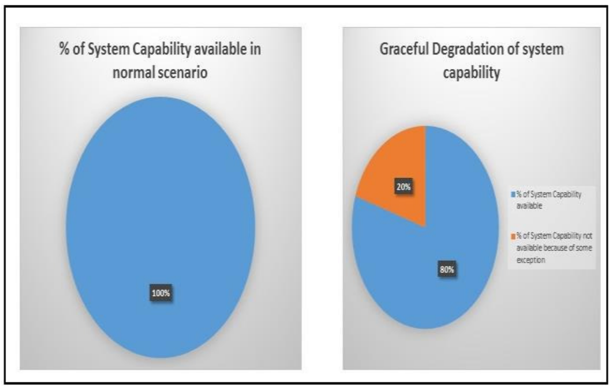

Figure 1 : Graceful Degradation

II. What issues Graceful Degradation solve

Graceful Degradation design can help solve multiple issues associated with unexpected events like an error/failure scenarios and how the system behaves and interacts with end-users in such scenarios. Some of the issues it addresses are

- Avoids high costs associated with setting up failover/HA instances of all service components, instead only most critical flows/service paths are supported
- System can still remain responsive , albeit with a lower state of services it offers
- System can communicate with end users, providing them updates on the failures and limited functionality, which still can be carried out

Note that Graceful degradation design doesn’t rule out other requirements and methods to achieve a fault tolerant system, e.g. having a DR site etc., but adds on to these features. With the help of Graceful degradation design, a smaller subset of services are still continued to be offered from the main site or DR site. This paper doesn’t necessarily differentiate between these aspects/instances etc., unless stated otherwise.

III. Handling Graceful Degradation

Graceful Degradation system design can vary from system to system. But at high level, common steps or patterns can be identified, which are critical for a system to degrade gracefully in case of an exception scenario. Figure 2 present a high level design approach of building graceful degradation in the system. Some of the important aspects of such systems are considered and explained how these can be used to achieve graceful degradation. High level flow to achieve graceful degradation can be summarized as follow

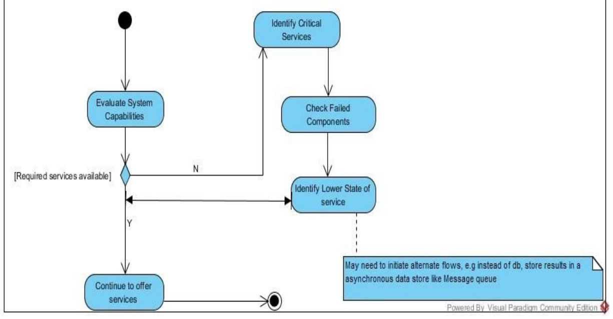
Identify Critical Services
Evaluate System Capabilities
Check Failed Components
[Required services available]
N
Identify Lower State of service
Y
May need to initiate alternate flows, e.g instead of db, store results in a asynchronous data store like Message queue
Continue to offer services
Powered By Visual Paradigm Community Edition

Figure 2 High Level Flow of Graceful Degradation

- System Analyzes and evaluates if all capabilities are fully functional
- If all required capabilities are available, continue to offer a full set of services to end consumers
- If all required capabilities are not available, identify the critical services, which should continue to be offered (with degraded performance/functionality)
- Check the components which have failed capabilities, e.g. it can check whether database or other subcomponent is available and working fine or not (based on predefined health check routine, response times, response codes, scheduled outages etc.)
- Once failed components are identified, evaluate and identify the set of services, which can continue to be offered with degraded set of functionality
- Note that to offer degraded set of services, some alternate/exception flows may need to be invoked for individual service/use case.
- For example, if some database service is not available, for query, data can be referenced from local cache or for writes, records/transactions can be written to some temporary message stores like message queues etc. Once database services are restored, these transaction details can be updated in the db
- Note that, a system handling graceful degradation, will also have a check to see if services are back up and running again. As more and more components start coming up, more and more services will continue to be offered to the end consumer.
- Also, note that in some scenarios, offering a subset of services can have cascading effect on the overall system. E.g. offering only one subset of functionality can have an impact on other service’s functionality, which in turn can impact another set of services. This can bring down the whole system. This scenario is very common in today’s business applications, where one service is integrated with multiple services to achieve a common business goal.

IV. Real life implementation of Graceful Degradation

In this section, we will see how Graceful Degradation can be built into the design of a telecom company’s IT applications. Consider an imaginary telecom operator “UserComm Limited”, which has a huge subscriber base of Prepaid and Postpaid customers. In this paper, we will focus on prepaid subscribers and applications related to prepaid subscribers. Figure 3 shows the high level context diagram of some of the most commonly used applications by a prepaid subscriber. For simplicity, we will not be discussing other applications, which may be there in IT eco-system of a telecom operator.

As shown in the context diagram in Figure 3, a prepaid subscriber can use the following set of applications to perform various operations:

- Use Prepaid Balance Enquiry service to check the current balance/talktime available
- Perform a top-up/recharge to get additional talktime
- Perform various operations like enrolling for new services/user profile management, setting preferred language etc. using SelfCare channel application
- And get in touch with Customer care over web/call etc. to solve certain service related issues/inquiries

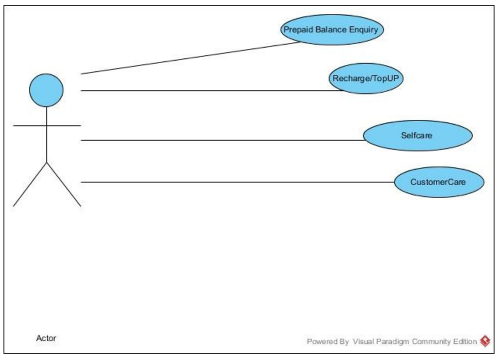

Figure 3 Context Diagram for a Prepaid Mobile IT system

Note that it may happen that in certain scenarios, it may be decided to completely stop offering one or more services, while continuing to offer other services. That is also an aspect of graceful degradation. But in this section, we will talk about how, for a particular application, small set of functionality can be offered to handle graceful degradation. The techniques presented can be used to cover the first scenario also.

The rest of this paper explains how graceful degradation aspect can be built into one of the applications shown above. This paper will talk about one of the service 'Prepaid Balance Enquiry' in detail, and also discuss how this service can be designed to handle graceful degradation in the event of exception/outages caused by various factors.

V. Working of Prepaid Balance Enquiry service

Figure 4 shows the high level working of Prepaid Balance Enquiry Service. Note that this is a simplified working of the actual service and many details are omitted to avoid complexity and to keep the focus only on the graceful degradation design aspect.

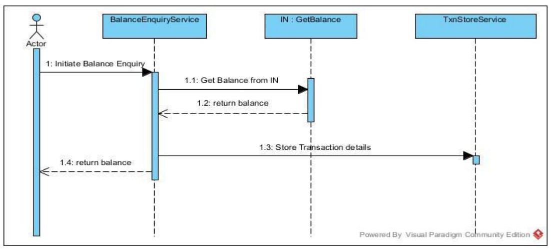

Figure 4 Sequence diagram of Prepaid Balance Enquiry service

Working of Prepaid Balance Enquiry can be summarized as:

- Prepaid subscriber sends request to get his balance
- Once request is received by the BalanceEnquiry Service, after performing necessary validations/authentication (not shows in the sequence diagram to avoid unnecessary complexity) , balance is retrieved from system knows as Intelligent Network(IN)
- IN system in a telecom environment are used for a wide variety of functions, one of which is to maintain the current balance of a prepaid subscriber (details of IN system can be found in the links given in the reference section)
- Once balance is retrieved from the IN system, the transaction details are stored in database , with the help of TxnStoreService
- Transaction details needed to be stored for various reporting, analytics and regulatory requirements.

VI. Working of Prepaid Balance Enquiry service

There are some high level issues in the above mentioned working of Prepaid Balance Enquiry service w.r.t. availability and reliability aspects.

- IN can be a single point of failure .In case IN is not available, balance cannot be retrieved
- In case of database services are not available, TxnStoreService will not be able to store the transaction details, thus there can be a challenge in meeting reporting , analytics and regulatory requirements

VII. Handling graceful degradation of Prepaid Balance Enquiry service in case of failure of IN system instances with 'Capability Determination Model' (CDM)

To handle the failures of IN system instances gracefully, a 'Capability Determination Model' (CDM) can be used. Figure 5 summarizes, how CDM can be used to handle IN system related exception scenarios.

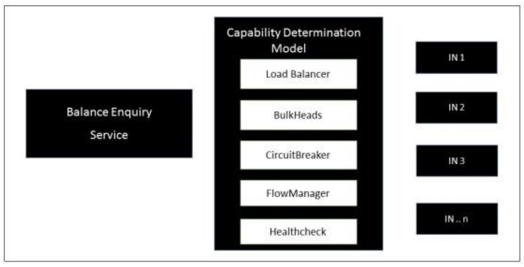

Figure 5 Model to gracefully degradation of services being offered in case of IN failure

Working of 'Capability Determination Model' (CDM), as shown in figure 5, can be explained as follows:

- To avoid IN as single point of failure, and also because IN systems are used by multiple applications in a telecom environment, sufficient redundancy is generally built for a given IN instance, by having active-active nodes working in tandem to serve requests (shown as IN1, IN2 and so on in the figure 5)
- CDM will use the services of Load Balancer to round robin the requests to the set of IN servers
- Concurrency to a particular IN server is throttled by use of Bulkheads. Bulkhead is a dedicated connection pool to individual IN server, using which max concurrent hits on a given IN server can be controlled. It can help in avoiding the overloading of any given IN server instance.
- In case an IN server instance is not available, because of some outages etc., Circuit breaker will mark that instance of the IN server as down, and will remove it from the Bulkhead and load balancer lists, so that no requests can be sent to the 'out of service' instance
- The FlowManager component of CDM will control the flows in various scenarios. For example, in case of none of the active IN instance is available, FlowManager will activate the alternate flow to handle this scenario
- Finally, there will be a Healthcheck module, which will keep on monitoring different IN server instances, and inform circuit breaker if any IN server instance is down or if any IN instance , which was not available earlier, has come up again

Figure 6 explains, how services offered are gracefully degraded for Prepaid balance enquiry service, in the scenario of one or more IN server instance going down because of expected or unexpected reasons.

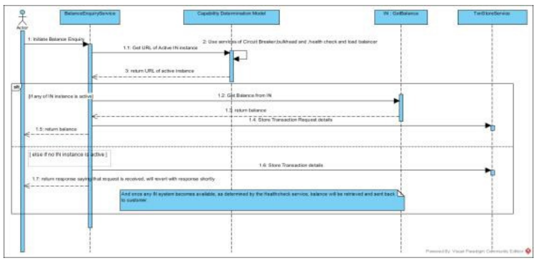
Actor
BalanceEnquiryService
Capability Determination Model
IN : GetBalance
TxnStoreService
1: Initiate Balance Enquiry
1.1: Get URL of Active IN instance
2: Use services of Circuit Breaker,bulkhead and ,health check and load balancer
3: return URL of active instance
alt
[If any of IN instance is active]
1.2: Get Balance from IN
1.3: return balance
1.4: Store Transaction Request details
1.5: return balance
[else if no IN instance is active]
1.6: Store Transaction details
1.7: return response saying that request is received, will revert with response shortly
And once any IN system becomes available, as determined by the Healthcheck service, balance will be retrieved and sent back to customer

Figure 6 Sequence diagram of graceful degradation of services offered by Prepaid balance enquiry

Above sequence diagram of the flow of working of Prepaid balance enquiry service, and how it handles the exception scenario gracefully and lowers down the level of services it offers, is explained below.

- Prepaid subscriber sends request to get his balance
- Once request is received by the BalanceEnquiry Service, after performing necessary validations/authentication (not shows in the sequence diagram to avoid unnecessary complexity) , it gets the URL of active instance of the IN system from 'Capability Determination Model'(CDM)
- 'Capability Determination Model' (CDM) will use the services of circuit breakers, bulkheads modules to figure out the active IN server instance and to get hold of a connection to one of the active IN instance using Load Balancer.
- If an active instance of IN server is available, balance will be retrieved, transaction details stored in the datastore and the balance will be returned to the subscriber. This behavior is exactly like the flow in figure 4, as this is the happy path.
- The only difference is that here IN is no longer a single point of failure and a suitable redundancy is built for IN server instances, by providing a set of active-active instances
- In case none of the instance of the IN server is available, then there will not be any URL available for serving the request
- CDM will determine this with the help of circuit breakers and bulk heads, as explained in description of figure 5.
- Once it is observed that no active instance of the IN server is available to serve the request, CDM will also trigger an alternate flow,with the help of FlowManager component, which will be offering degraded services
- Now, instead of sending a balance amount as a response to the customer in real time, request details would be stored in the transactional datastore
- A response to subscriber will be sent that "his request has been received and the system will revert with the details".
- This ensures that system continues to be responsive and keeps on communicating with users, instead of dying out completely
- Once IN server instance is available, all the pending requests can be retrieved from transactional data store , processed and final response sent to the subscriber
- Thus, in the case failure of IN instances ,system was still able to accept requests, and communicate with end user, thus providing better user experience and satisfaction
- Although, the system capability was degraded from 'sharing the current balance in real time' to 'sharing the current balance after some time window (offline mode)', still system was able to perform the services.
- Similarly, in a scenario, where one out of two IN server instances was down, the system should still be able to handle 50% requests in real time and the rest of the requests in offline mode

Table 1 summarizes the scenarios of the failures of the IN server instances, how much system functionality or capability is degraded, and how system degrades gracefully to handle the requests in offline mode.

Table 1 Summary of graceful degradation from real time to offline fulfillment of requests

Sr. No	Component Name	# of Instances Available	Mode of Services offered	Capability	Comment
1	IN Instance	3	Real Time	100%	System working with full capacity in real time
2	IN Instance	2	Real Time and offline	66 % real time, 33 % offline	33 % requests will be handled offline , after graceful degradation to offline mode
3	IN Instance	1	Real Time and offline	33 % real time, 66 % offline	66 % requests will be handled offline , after graceful degradation to offline mode
4	IN Instance	0	Offline Mode	100 % offline	100 % requests will be handled offline , after graceful degradation to offline mode

VIII. Handling graceful degradation of Prepaid Balance Enquiry service in case of failure of database services with 'Capability Determination Model' (CDM)

To handle the failures of database services gracefully, 'Capability Determination Model' (CDM) can be used. Figure 7 summarizes, how CDM can be used to handle

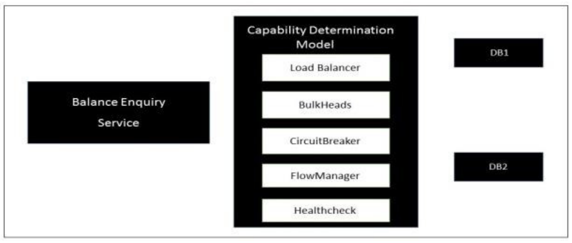

Figure 7 Model to gracefully degradation of services being offered in case of Database failure

database services related exception scenarios. Working of 'Capability Determination Model' (CDM) as shown in figure 7, can be explained as follows:

- To avoid the database as single point of failure, , sufficient redundancy is generally built for a given database instance, by having active-active nodes working in tandem to serve requests (shown as DB1 , DB2 in the figure 5)
- CDM will use the services of Load Balancer to round robin the requests to the set of DB servers
- Concurrency to a particular Database server is throttled by use of Bulkheads. Bulkhead is a dedicated connection pool to individual database server, using which max concurrent hits on a given database server can be controlled. It can help in avoiding the overloading of any given database server instance.
- In case a database server instance is not available, because of some outages etc., Circuit breaker will mark that instance of the database server as down, and will remove it from the Bulkhead and load balancer lists, so that no requests can be sent to the 'out of service' instance
- The FlowManager component of CDM will control the flows in various scenarios. For example, in case of none of the active database instance is available, FlowManager will activate the alternate flow to handle this scenario
- Finally, there will be a Healthcheck module, which will keep on monitoring different database server instances, and inform circuit breaker if any database server instance is down or if any database instance , which was not available earlier, has come up again

Figure 8 explains, how services offered are gracefully degraded for Prepaid balance enquiry service, in the scenario of one or more database server instances going down because of expected or unexpected reasons.

Note that for simplicity, following sequence diagram only shows the database operation of storing transaction details in the database store. Other operations like getting balance from the IN servers and how graceful degradation is handled in those scenarios have been omitted from this figure, as these have already been explained in detail, in the last section.

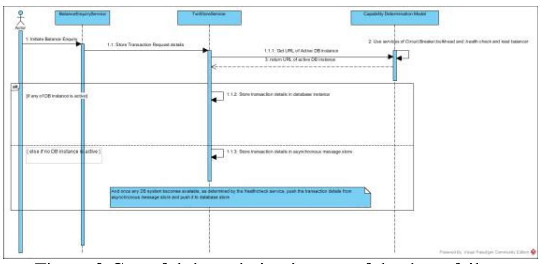
Actor
BalanceEnquiryService
TxnStoreService
Capability Determination Model
1: Initiate Balance Enquiry
1.1: Store Transaction Request details
1.1.1: Get URL of Active DB instance
2: Use services of Circuit Breaker,bulkhead and ,health check and load balancer
3: return URL of active DB instance
alt
[If any of DB instance is active]
1.1.2: Store transaction details in database instance
[else if no DB instance is active]
1.1.3: Store transaction details in asynchronous message store
And once any DB system becomes available, as determined by the Healthcheck service, push the transaction details from asynchronous message store and push it to database store
Powered By Visual Paradigm Community Edition

Figure 8 Graceful degradation in case of database failure

Above sequence diagram (Figure 8) of the flow of working of database operations of Prepaid balance enquiry service, and how it handles the database server exception scenario gracefully and lowers down the level of services it offers, is explained below.

- Prepaid subscriber sends request to get his balance
- Once request is received by the BalanceEnquiry Service, after performing necessary validations/authentication (not shows in the sequence diagram to avoid unnecessary complexity), further steps are explained in detail, while explaining figure 6 in the last section.
- Once steps to get the balance from the IN system, as explained in figure 6, are completed, transaction details need to be stored in database server (in case of none of the active IN server instance is available, as explained in figure 6, requests details are also stored in the database. Hence, next set of steps are also applicable to this scenario, when requests details are needed to be stored in the database)
- TxnStoreService , the service which handles the database related operations to store transaction details, get the URL of the active database instance from 'Capability Determination Model'(CDM)
- 'Capability Determination Model' (CDM) will use the services of circuit breakers, bulkheads modules to figure out the active database server instance and to get hold of a connection to one of the active database instance using Load Balancer.
- If an active instance of database server is available, transaction details stored in the datastore and the balance returned to the subscriber. This behavior is exactly like the flow in figure 4, as this is the happy path.
- Only difference is that here database is no longer a single point of failure and a suitable redundancy is built for database server instances, by providing a set of active-active instances
- In case none of the instance of the database server is available, then there will not be any URL available for serving the request
- CDM will determine this with the help of circuit breakers and bulkheads, as explained in description of figure 5.
- Once it is observed that no active instance of the database server is available to store the transaction details , CDM will also trigger an alternate flow, which will be offering degraded services
- Now, instead of storing the transaction details in the database, all requests will be stored in a asynchronous messaging system like ActiveMQ or IBM WebSphere MQ
- Similarly, read operations can happen from a cached copy of some of the database tables, or from a read only replica of the database
- This way, the system will be able to continue with its working, but reduced and degraded capabilities like analytical and reporting might now happen on this new set of data, till the time the database server becomes active again and transaction details in the messaging system are committed to the database server
- This ensures that system continues to be responsive and keeps on communicating with users, instead of dying out completely
- Once the database server instance is available, all the transaction details can be retrieved from messaging system and pushed to the transactional data store
- Although, the system capability was degraded from 'sharing the current balance in real time' to 'sharing the current balance after some time window (offline mode)', or 'not able to support analytical and reporting requirements' , in case all database instances were down, still system was able to perform the other critical services.

Table 2 summarizes the scenarios of the failures of the database server instances, how much system functionality or capability is degraded, and how system degrades gracefully to handle the requests in offline mode, or stops offering analytical and reporting requirements, in case none of the database instance is available.

Table 2 Summary of graceful degradation (real time & offline mode, reporting & analytical requirements)

Sr. No	**Component Name**	**# of Instances Available**	**Mode of Services offered**	**Capability**	**Comment**
1	DB Instance	2	Real Time	100%	System working with full capacity in real time
2	DB Instance	1	Real Time and offline	50 % real time, 50 % offline	50 % requests will be handled offline , after graceful degradation to offline mode
3	DB Instance	0	Offline Mode	100% offline	100 % requests will be handled offline, after graceful degradation to offline mode. Analytical and reporting services also gracefully downgraded and no longer will be available, till database server comes up again.

IX. High level working of components of Capability Determination Model (CDM)

This section discusses about the high level working of critical components of the proposed CDM. It will give very good idea and insights into the working of these components.

A. *Working of BulkHeads*

Figure 9, summarizes the high level working of bulkheads. Critical details of the working of bulkheads are also explained in this section.

- Any application, which needs to connect to a component

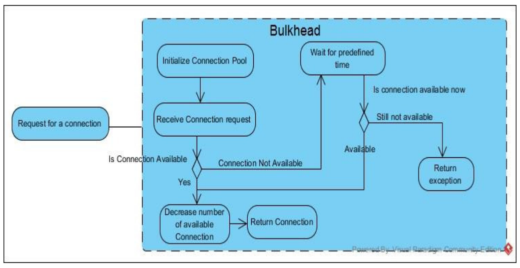

Figure 9 Working of Bulkhead component of CDM

(say database or IN server instance, as per scenario explained in this paper), will request for the connection to the component
- Connection to each of the components is controlled by individual bulkhead
- During startup, each of the bulkhead will initialize the respective connection pool
- When a request for a new connection is made, the bulkhead will check if the connection to the requested component is available to serve the request
- If the connection is available, it will allocate this connection to serve the request and decrease the number of active connections by one. This is needed to keep track of free and allocated connections
- In case, no free connection is available, bulkhead will wait for a pre-defined time interval.
- If any connection becomes available during this wait period, it will be allocated to serve the requests, else an exception can be returned , indicating no connection is available

B. *Working of CircuitBreaker*

Figure 10, summarizes the high level working of circuit breakers. Critical details of the working of circuit breakers are also explained in this section.

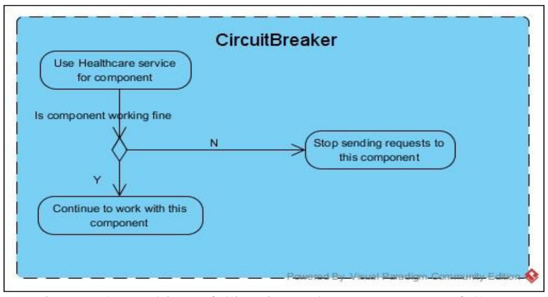
CircuitBreaker
Use Healthcare service for component
Is component working fine
N
Stop sending requests to this component
Y
Continue to work with this component
Powered By Visual Paradigm Community Edition

Figure 10 Working of CircuitBreaker component of CDM

- Circuit-breaker will use the service of the Healthcheck component, to see if a given component is working fine
- If Healthcheck of this component is turned out to be fine, based on policies defined in Healthcheck component, circuit breaker will continue to work with this component by sending the requests to this component, with the help of bulkheads
- In case, this component is not working fine, based on policies defined in Healthcheck component, the circuit breaker will stop sending requests to this component all together

C. *Working of Healthcheck*

Figure 11 summarizes the high level working of Healthcheck. Critical details of the working of Healthcheck is also explained in this section.

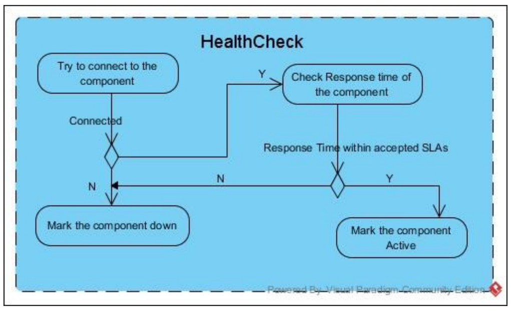
HealthCheck
Try to connect to the component
Y
Check Response time of the component
Connected
Response Time within accepted SLAs
N
N
Y
Mark the component down
Mark the component Active
Powered By Visual Paradigm Community Edition

Figure 11 Working of HealthCheck component of CDM

- Healthcheck component will try to connect to each instance of the component, defined in external configurations of Healthcheck component
- In case connection is not successful, Healthcheck component will mark this instance of the component as down
- Any instance, which will be marked down by the Healthcheck component, Bulkheads will not try to connect to that instance of the component
- When all the instances of a component are marked down by the Healthcheck component, CircuitBreaker will come into action and will stop accessing the component all together
- CDM's Flow manager will kick in, and initiate an alternate flow, in the event of all instances of a component are marked down
- However, if the connection to the instance of component was successful, the Healthcheck component will also check the response time from this instance
- If response time from this instance was beyond SLAs, even in this case, instance will be marked as down
- If the response time and other policies, defined in the Healthcheck component are met, this instance will be marked as active, and will continue to be in use by Bulkheads

D. *Working of Flow Manager*

Figure 12 summarizes the high level working of Flow Manager. Critical details of the working of Workflow Manager is also explained in this section.

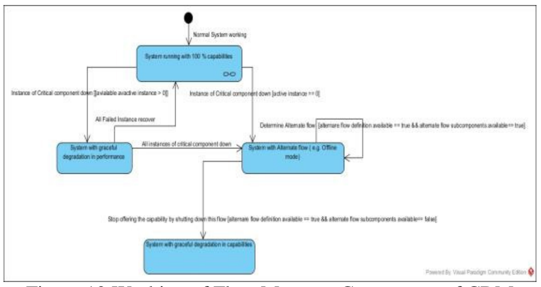
Normal System working
System running with 100 % capabilities
Instance of Critical component down [available avactive instance > 0]
Instance of Critical component down [active instance == 0]
All Failed Instance recover
Determine Alternate flow [alternare flow definition available == true && alternate flow subcomponents available== true]
System with graceful degradation in performance
All instances of critical component down
System with Alternate flow (e.g. Offline mode)
Stop offering the capability by shutting down this flow [alternare flow definition available == true && alternate flow subcomponents available== false]
System with graceful degradation in capabilities
Powered By Visual Paradigm Community Edition

Figure 12 Working of Flow Manager Component of CDM

- ➢ Initially, the system is in a normal working state with full capabilities on offer
- ➢ During the execution cycle, one or more instances of one of the critical component go down(e.g. one of the database instance)
- ➢ If one or more active instance of the critical component is still available, the system will continue to offer the full capabilities , but the graceful degradation in performance (e.g. now instead of two instances, only one of the database instance is available and handling load)
- ➢ In case, none of the instance of the critical component is active, workflow manager, with the help of configured flow definition, will initiate the alternate flow (e.g. switching the flow from online to offline mode, by storing the transaction details in a message oriented middleware)
- ➢ Thus the system will shut down the capability, and start offering some of the gracefully degraded capabilities with the help of alternate flows.
- ➢ Note that there can be additional state transitions as well, but not shown here for simplicity. These transitions could be the system with degraded performance switching back to normal state, in case the instance which was marked down, comes up again, or after all components are available, the system starts offering the complete functionalities , by switching to fully normal working state

X. Further research and design improvements in Capability Determination Model (CDM)

The proposed design and working of CDM can be improved further by doing more research around the following areas:

- ➢ One of the emerging software architecture pattern "Microservices based software system architecture" has inherent quality of "low coupling" between different components. This feature of microservices based system can allow simpler and much cleaner graceful degradation. So, microservice architecture should be explored further to identify design approaches for handling graceful degradation in microservices based software system
- ➢ Currently, CDM and all the building blocks of CDM like circuit-breakers and bulkheads are reactive in nature
- ➢ It means, these components only detect and take action for the failures and exception scenarios, which have already occurred
- ➢ CDM can be further improved by having a component called 'CorrelationManager',which will correlate all the historical failure and exception scenarios, determine the causes and outcome/impact of all such failure events
- ➢ It will also perform predictive analytics on this historical data (collected from application and server logs etc.) and determine when the next failure event is likely to occur
- ➢ Similarly, planned outage details like rollout, maintenance and upgrade schedules can be fed to this new component of CDM
- ➢ Based on the outcome of the predictive analytics on the historical failure scenario logs and planned outage details, CDM can trigger the alternate flows pro-actively, without any manual intervention
- ➢ Thus, the systems can be more pro-active in predicting and handling failure scenarios, instead of taking corrective actions, only after the failure event has already occurred

XI. Alternatives of Graceful degradation system design

Table 3 lists down a couple of alternatives to graceful degradation system design. Both the advantages and disadvantages of each of this approach are also discussed.

Table 3 Advantages and disadvantages of alternatives of Graceful Degradation

Sr. No.	Alternative Approach	Advantages	Disadvantages
1	Not handling exception and failure scenarios gracefully (no capability degradation, no failover etc.)	➢ No extra effort required to design and implement ➢ Costs of designing and setting up	➢ One exception/ failure event can cause complete system outage ➢ Cost of outages can

		alternate flows can be avoided	be very high to business ➢ User experience can be impacted because of non-responsive system
2	Providing failover/alternate instances of all subcomponents of the system, to handle failure scenarios	➢ Provides 'always available' system ➢ Complete system functionality and capabilities will always be available, even in case of a disaster	➢ Setting up failover and alternate instances of all subcomponent can be a costly effort ➢ In today's world of integrated application environment, providing failover for all sub-components may not be feasible, as some applications may lie outside the scope of a single organization, e.g. vendor applications, COTS etc.

XII. Challenges in Designing Software Systems with Graceful Degradation

Designing and implementing software systems with graceful degradation has its own challenges. But the benefits it provides, in terms of availability and fault tolerance of critical services, it is an effort worth taking to try to mitigate the challenges of designing such systems. Some of the challenges in graceful degradation system design are listed below

- ➢ Determining the impact of an failure scenario in complex business flows can be difficult
- ➢ Failures in any part of the system, with multiple integrations, can have cascading effect on the working of overall system. Determining the services capabilities , which can still function, can be a challenging task
- ➢ Extra effort is needed to design and implement strategies for handling graceful degradation , which can have impact on the project budget, in terms of cost and schedule

XIII. Conclusion

Handling exception and failure scenarios gracefully is an aspect of IT system, which if handled well, can increase the system availability and user experience. By handling such scenarios gracefully and lowering the capabilities the system offer, in case of failure scenarios, is called graceful degradation. This paper talked about the concept of graceful degradation, and presented a high level approach of how graceful degradation can be designed into a system. This paper further elaborated how graceful degradation was designed and implemented in a real world application for a telecom operator. It talked about the "Capability Determination Model" (CDM), which can be designed and built to handle couple of such failure scenarios and gracefully degrade the system capabilities.
In the end, this paper also presented the high level approach of working of components of CDM. This approach can be used to build and design such models in other IT systems as well. Finally, this paper talked about some of the challenges, which can make designing systems with graceful degradation, difficult.

Acknowledgment

Sincere thanks to Tarun Sukhu (tsukhu@hcl.com) and Suneev Motwani(mot@hcl.com) for reviewing this paper and sharing valuable inputs.

References

[1] Titos Saridakis . Design Patterns for Graceful Degradation - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.996&rep=rep1&type=pdf

[2] Towards Robust Distributed Systems , http://www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[3] Rad aideh, Moh'd A.. Architecture of Reliable Web Applications Software . http://www.igi-global.com/book/architecture-reliable-web-applications-software/78

[4] Circuit Breaker Design Pattern - https://en.wikipedia.org/wiki/Circuit_breaker_design_pattern

[5] Bulkhead design Pattern - http://stackoverflow.com/questions/30391809/what-is-bulkhead-pattern-used-by-hystrix

[6] ActiveMQ - http://activemq.apache.org/

[7] IBM WebSphere MQ - http://www-03.ibm.com/software/products/en/ibm-mq