TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────

ĐỒ ÁN

**TỐT NGHIỆP ĐẠI HỌC**

NGÀNH CÔNG NGHỆ THÔNG TIN

**TÊN ĐỀ TÀI**

Hệ thống game online trên di động ứng dụng kĩ thuật phần mềm chịu lỗi

Sinh viên thực hiện : **Nguyễn Văn Thịnh**

Lớp CNPM – K52

Giáo viên hướng dẫn: ThS **Lê Đức Trung**

HÀ NỘI 06-2012

# PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

**1. Thông tin về sinh viên**

Họ và tên sinh viên: Nguyễn Văn Thịnh

Điện thoại liên lạc : 01257523333 Email: vanthinh061189@gmail.com

Lớp: Công nghệ phần mềm K52 Hệ đào tạo: Chính quy

Đồ án tốt nghiệp được thực hiện tại: Đại học Bách Khoa Hà Nội

Thời gian làm ĐATN: Từ ngày 27/02/2012 đến 31/05/2012

**2. Mục đích nội dung của ĐATN**

Xây dựng hệ thống game online trên các thiết bị di động hỗ trợ đa nền tảng (J2ME, Android, … ) theo mô hình client – server có áp dụng các lý thuyết về xây dựng phần mềm chịu lỗi. Thiết kế một framework chung cho việc phát triển game online trên di động. Nghiên cứu về lý thuyết xây dựng phần mềm chịu lỗi và áp dụng vào hệ thống game online để đảm bảo hệ thống vẫn có khả năng cung cấp các dịch vụ khi xảy ra lỗi.

**3. Các nhiệm vụ cụ thể của ĐATN**

- Tìm hiểu đề tài và xây dựng đề cương.

- Thiết kế kiến trúc client – server cho hệ thống game online.

- Tìm hiểu lý thuyết về phần mềm chịu lỗi.

- Thiết kế mô đun chịu lỗi cho một số thành phần quan trọng của hệ thống.

- Lập trình client, server và mô đun chịu lỗi.

- Hoàn thiện báo cáo và tài liệu của đồ án.

**4. Lời cam đoan của sinh viên:**

Tôi – *Nguyễn Văn Thịnh* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *ThS Lê Tấn Hùng và ThS Lê Đức Trung*.

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

|  |  |
| --- | --- |
|  | *Hà Nội, ngày 30 tháng 5 năm2012*  Tác giả ĐATN  *Nguyễn Văn Thịnh* |

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

|  |  |
| --- | --- |
|  | *Hà Nội, ngày 30 tháng 5 năm 2012*  Giáo viên hướng dẫn  *ThS Lê Đức Trung* |

# TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP

Hiện nay với sự phát triển nhanh chóng của khoa học công nghệ, các thiết bị điện thoại di động ngày càng trở nên phổ biến đặc biệt là các dòng smartphone cao cấp sử dụng các hệ điều hành mới ( Android, iOS, … ) đang phát triển nhanh chóng. Chiếc điện thoại hiện tại không chỉ được sử dụng với chức năng nghe gọi thông thường mà còn được sử dụng rất nhiều với mục đích giải trí như chơi game, nghe nhạc, …. Nhận thấy cơ hội trong việc phát triển các ứng dụng giải trí trên di động em đã lựa chọn đề tài đồ án tốt nghiệp: *“Xây dựng hệ thống Game Online trên di động ứng dụng các kĩ thuật chịu lỗi“* . Mục tiêu mà đề tài này hướng tới là xây dựng thành công một hệ thống server game có khả năng đáp ứng tốt cho nhu cầu chơi game online trên các thiết bị di động. Điều này đòi hỏi người thực hiện không chỉ tập trung nghiên cứu và xây dựng hệ thống về phía server mà còn phải có sự hiểu biết sâu rộng về lập trình phần mềm trên các thiết bị di động nói chung và lập trình game trên mobile nói riêng. Bên cạnh đó để nâng cao chất lượng và độ tin cậy của hệ thống em đã nghiên cứu về lý thuyết chịu lỗi để áp dụng vào hệ thống.

Trong quá trình thực hiện đề tài em đã tìm hiểu về cách thức xây dựng một server game, xây dựng kiến trúc server–client cho game online trên di động, tìm hiểu về giao thức truyền tin giữa client–server và cách thức xây dựng game trên thiết bị di động. Bên cạnh đó em cũng tập trung nghiên cứu lý thuyết và các kĩ thuật chịu lỗi thường được áp dụng trong các mô hình phần mềm hiện tại để lựa chọn ra một kĩ thuật phù hợp nhất với hệ thống của mình. Khi áp dụng các kĩ thuật chịu lỗi, hệ thống sẽ vẫn đảm bảo đáp ứng các dịch vụ cần thiết đến người sử dụng khi xảy ra lỗi (về cả phía client và server).

Dựa trên việc xác định được mục tiêu và nhiệm vụ mà đề tài cần đạt được, em đã thực hiện đồ án tốt nghiệp của mình với bố cục được trình bày như sau :

* Chương 1: Đặt vấn đề và định hướng giải pháp. Trong chương này em sẽ mô tả chi tiết bài toán đã đặt ra, phân tích các vấn đề cần giải quyết và đề ra các giải pháp để giải quyết các vấn đề.
* Chương 2: Lý thuyết về phần mềm chịu lỗi. Chương này có nhiệm vụ trình bày các lý thuyết và các kĩ thuật cơ bản được sử dụng trong việc thiết kế một hệ thống có khả năng chịu lỗi.
* Chương 3: Phân tích kiến trúc hệ thống. Phân tích và đề ra được một kiến trúc phù hợp với hệ thống game online trên thiết bị di động. Đồng thời tiến hành phân tích các thành phần dễ xảy ra lỗi để thiết kế các kĩ thuật chịu lỗi đạt hiệu quả cao nhất.
* Chương 4: Thiết kế hệ thống. Dựa vào kiến trúc hệ thống đã phân tích ở trên tiến hành thiết kế các thành phần của hệ thống (server, client, …)
* Chương 5: Cài đặt và triển khai hệ thống. Trình bày về quá trình cài đặt hệ thống. Các kết quả thu được khi chạy thử nghiệm và triển khai hệ thống trên các thiết bị thật.
* Chương 6: Kết luận. Đánh giá những điểm đã làm được và những thiếu xót cần phải khắc phục. Đề ra những giải pháp để có thể cải thiện và nâng cao chất lượng của hệ thống.

Trong quá trình thực hiện đồ án em sẽ xây dựng một hệ thống game online hỗ trợ hai nền tảng di động phổ biến hiện nay là J2ME và Android từ đó sẽ đánh giá được một cách toàn diện những ưu, nhược điểm của hệ thống mà em đã thiết kế và xây dựng.

# LỜI CẢM ƠN

Lời đầu tiên, em xin được gửi lời cảm ơn chân thành đến các thầy cô giáo trong trường Đại học Bách Khoa Hà Nội cũng như các thầy cô giáo trong Viện Công nghệ thông tin và Truyền thông đã truyền dạy cho em những kiến thức và kinh nghiệm quý giá trong suốt quá trình học tập tu dưỡng trong 5 năm qua.

Đặc biệt, em xin gửi lời cảm ơn đến thầy Lê Tấn Hùng và thầy Lê Đức Trung, Viện Công nghệ thông tin và truyền thông, các thầy đã hướng dẫn và chỉ bảo em trong suốt 4 năm chuyên ngành và toàn bộ quá trình làm đồ án tốt nghiệp.

Em cũng muốn gửi lời cảm ơn chân thành đến tập thể lớp Công nghệ phần mềm K52 đã tạo môi trường thi đua học tập lành mạnh, tạo điều kiện cho sự phát triển của các thành viên trong lớp.

Cuối cùng, em xin được gửi lời cảm ơn chân thành tới gia đình, bạn bè đã quan tâm, động viên, đóng góp ý kiến và giúp đỡ em trong quá trình học tập, nghiên cứu và hoàn thành đồ án tốt nghiệp.

Hà Nội, ngày 19 tháng 05 năm 2011

Nguyễn Văn Thịnh

Lớp CNPM – K52

# MỤC LỤC

[PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP 2](#_Toc325313379)

[TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP 3](#_Toc325313380)

[LỜI CẢM ƠN 5](#_Toc325313381)

[MỤC LỤC 6](#_Toc325313382)

[CHƯƠNG 1. ĐẶT VẤN ĐỀ VÀ ĐỊNH HƯỚNG GIẢI PHÁP 8](#_Toc325313383)

[1.1 Mô tả chi tiết đề tài 8](#_Toc325313384)

[1.2 Các vấn đề và định hướng giải pháp 8](#_Toc325313385)

[CHƯƠNG 2. LÝ THUYẾT PHẦN MỀM CHỊU LỖI 11](#_Toc325313386)

[2.1 Khái niệm chung về phần mềm chịu lỗi 11](#_Toc325313387)

[2.1.1 Phần mềm chịu lỗi 11](#_Toc325313388)

[2.1.2 Phân loại lỗi 11](#_Toc325313389)

[2.1.3 Lý thuyết độ tin cậy 12](#_Toc325313390)

[2.1.4 Lý thuyết dự phòng (Redundancy) 18](#_Toc325313391)

[2.2 Các kĩ thuật xây dựng phần mềm chịu lỗi 20](#_Toc325313392)

[2.2.1 Kĩ thuật phần mềm chịu lỗi cho một phiên bản phần mềm 20](#_Toc325313393)

[2.2.2 Kĩ thuật phần mềm chịu lỗi cho nhiều phiên bản phần mềm 23](#_Toc325313394)

[CHƯƠNG 3. Xây dựng mô hình Master-Slave-Client 33](#_Toc325313395)

[3.1 Mô hình kết nối client – server sử dụng socket 33](#_Toc325313396)

[3.1.1 Kết nối theo TCP 33](#_Toc325313397)

[3.1.2 Kết nối theo UDP 36](#_Toc325313398)

[3.2 Xây dựng server sử dụng socket trong Java 38](#_Toc325313399)

[3.3 Kiến trúc server 39](#_Toc325313400)

[3.3.1 Mô đun quản lý server 39](#_Toc325313401)

[3.3.2 Mô đun quản lý giao vận 40](#_Toc325313402)

[3.4 Kiến trúc client 41](#_Toc325313403)

[3.4.1 GUI – Giao diện người dùng 42](#_Toc325313404)

[3.4.2 Connection Handler 42](#_Toc325313405)

[3.5 Kĩ thuật chịu lỗi áp dụng cho mô đun kết nối 42](#_Toc325313406)

[3.5.1 Mô đun kết nối 42](#_Toc325313407)

[3.5.2 Áp dụng kĩ thuật chịu lỗi Recovery Block 43](#_Toc325313408)

[CHƯƠNG 4. THIẾT KẾ HỆ THỐNG 44](#_Toc325313409)

[4.1 Giới thiệu về sản phẩm 44](#_Toc325313410)

[4.2 Các chức năng của game 44](#_Toc325313411)

[4.2.1 Các chức năng cơ bản 44](#_Toc325313412)

[4.2.2 Các chức năng trong bàn chơi 46](#_Toc325313413)

[4.3 Cơ sở dữ liệu 48](#_Toc325313414)

[4.4 Master Server 51](#_Toc325313415)

[4.4.1 Mô đun quản lý server 52](#_Toc325313416)

[4.4.2 Mô đun quản lý kết nối 54](#_Toc325313417)

[4.4.3 Mô đun thao tác và quản lý cơ sở dữ liệu 56](#_Toc325313422)

[4.5 Slave Server 57](#_Toc325313423)

[4.5.1 Mô đun quản lý server 57](#_Toc325313424)

[4.5.2 Mô đun quản lý kết nối 60](#_Toc325313425)

[4.5.3 Mô đun thao tác và quản lý cơ sở dữ liệu 63](#_Toc325313429)

[4.6 Client 63](#_Toc325313430)

[4.6.1 GameClient 64](#_Toc325313431)

[4.6.2 ConnectionHandler 65](#_Toc325313432)

[4.6.3. ReaderThread 65](#_Toc325313433)

[4.6.4 WriterThread 66](#_Toc325313434)

[4.6.5 Định dạng gói tin 66](#_Toc325313435)

[4.6.6 Mô hình cấu trúc các mô đun của client 67](#_Toc325313436)

[CHƯƠNG 5. CÀI ĐẶT VÀ TRIỂN KHAI HỆ THỐNG 68](#_Toc325313437)

[5. 1 Giao diện game 68](#_Toc325313438)

[5.1.1 Giao diện cơ bản của game trên di động 68](#_Toc325313439)

[5.1.2 Giao diện game Đấu trường trí tuệ 69](#_Toc325313440)

[5.2 Triển khai hệ thống 74](#_Toc325313441)

[5.2.1 Môi trường và công cụ sử dụng 74](#_Toc325313442)

[5.2.2 Cài đặt và cấu hình hệ thống 75](#_Toc325313443)

[5.2.3 Chạy thử nghiệm hệ thống 76](#_Toc325313444)

[KẾT LUẬN 77](#_Toc325313445)

[TÀI LIỆU THAM KHẢO 78](#_Toc325313446)

[DANH SÁCH HÌNH VẼ 79](#_Toc325313447)

[DANH SÁCH CÁC BẢNG 81](#_Toc325313448)

# CHƯƠNG 1. ĐẶT VẤN ĐỀ VÀ ĐỊNH HƯỚNG GIẢI PHÁP

## 1.1 Mô tả chi tiết đề tài

Đề tài được em lựa chọn thực hiện: *“Xây dựng hệ thống Game Online trên di động ứng dụng các kĩ thuật chịu lỗi“.* Mục tiêu mà đề tài hướng tới là xây dựng một game trên các thiết bị di động với chất lượng tốt cùng với đó là xây dựng một game server với độ ổn định cao để người chơi có thể kết nối và sử dụng các chức năng chơi game trực tuyến. Các thiết bị di động hiện tại đang phát triển với tốc độ rất nhanh với ba nền tảng chủ yếu là: J2ME, Android, iOS cùng với đó là nhu cầu chơi game và sử dụng các ứng dụng giải trí trên các thiết bị di động cũng ngày càng tăng. Với việc mạng 3G đã được triển khai rộng rãi, các thiết bị di động hiện nay cũng có khả năng kết nối mạng khá tốt nên việc chơi game online trên di động hiện tại cũng trở nên dễ dàng hơn. Đề tài em thực hiện cũng hướng tới mục tiêu là có thể thực hiện thành một game online hoàn chỉnh, phát hành ra thị trường để đáp ứng nhu cầu của người sử dụng chứ không chỉ dừng lại ở mức nghiên cứu và thử nghiệm.

Với một hệ thống game online việc duy trì kết nối giữa thiết bị (client) với server và đảm bảo server luôn hoạt động ổn định là hai vấn đề quan trọng nhất. Với server dù là một lỗi nhỏ nhất cũng có thể gây ra những sai sót khi vận hành, khi đó sẽ kéo theo những lỗi cho cả hệ thống. Khi server bị lỗi, các chức năng của người sử dụng sẽ không được đáp ứng một cách chính xác, nguy hiểm hơn có thể dẫn tới sai sót trong cơ sở dữ liệu, mất mát dữ liệu hoặc cả hệ thống ngừng hoạt động. Vì vậy việc xây dựng server có khả năng chịu lỗi là rất quan trọng và là yếu tố quan trọng quyết định độ tin cậy của hệ thống. Để có thể tăng khả năng chịu lỗi của server thì trong quá trình xây dựng em cũng nghiên cứu và áp dụng các kĩ thuật chịu lỗi phù hợp. Các kĩ thuật chịu lỗi sẽ được áp dụng ở những mô đun quan trọng và dễ có khả năng xảy ra lỗi chẳng hạn như mô đun đảm nhận việc kết nối giữa client và server.

## 1.2 Các vấn đề và định hướng giải pháp

Khi đã xác định được đề tài và mục tiêu của bài toán một vấn đề rất quan trọng đặt ra là lựa chọn một mô hình phù hợp để xây dựng hệ thống. Sau một thời gian tìm hiểu và tham khảo các game online khác hiện đang có em đã lựa chọn mô hình để xây dựng là mô hình client – server. Với mô hình này có các thành phần chính là: client, server, thành phần kết nối giữa client và server.

Khi thực hiện hệ thống với mô hình này em đã xác định một số vấn đề cần phải giải quyết như sau :

* Lựa chọn nền tảng phát triển trên các thiết bị di động.
* Xây dựng kiến trúc client – server đáp ứng nhiều người sử dụng.
* Mô đun kết nối giữa client – server.
* Cơ sở dữ liệu của hệ thống.
* Các kĩ thuật chịu lỗi.
* Triển khai và kiểm thử hệ thống.

Với vấn đề lựa chọn nền tảng phát triển trên các thiết bị di động, hiện tại có rất nhiều các nền tảng di động khác nhau nhưng phổ biến nhất trên thị trường hiện tại là ba nền tảng J2ME, Android, iOS. Để thực hiện đề tài của mình em đã lựa chọn hai nền tảng là J2ME và Android. J2ME là một nền tảng đã tồn tại khá lâu và hiện tại cũng không còn được ưa chuộng nhưng số lượng người dùng sử dụng các thiết bị có hỗ trợ nền tảng này vẫn rất lớn. Trong khi đó Android là một nền tảng mới và đang phát triển rất mạnh mẽ, các thiết bị smart phone sử dụng hệ điều hành Android đang bùng nổ. Việc phát triển các ứng dụng và game trên Android cũng khá dễ dàng vì Android cũng sử dụng ngôn ngữ lập trình Java đã quá phổ biến, các game trên các thiết bị sử dụng hệ điều hành Android cũng đang bùng nổ và được người sử dụng rất ưa chuộng.

Mô hình được em lựa chọn để xây dựng hệ thống là client – server. Vấn đề đặt ra ở đây là khi hệ thống có nhiều người sử dụng cần đảm bảo hệ thống có khả năng chịu tải, khi hệ thống cần mở rộng việc mở rộng đó cũng cần được thực hiện dễ dàng và không làm ảnh hưởng đến việc đến các thành phần đang hoạt động. Để giải quyết vấn đề này em sẽ xây dựng hệ thống theo kiến trúc master – slave – client. Theo đó hệ thống sẽ có rất nhiều các “slave server” (có thể gọi là các server con) có nhiệm vụ thực hiện các dịch vụ mà client yêu cầu trong game. Các slave server này được quả lý bằng các master server, master server sẽ theo dõi tình trạng của các slave server và thông báo tới người sử dụng, khi một slave server xảy ra lỗi master server sẽ thực hiện các chức năng để đảm bảo hệ thống vẫn hoạt động bình thường. Cả slave server và master server đều sử dụng chung một cơ sở dữ liệu người dùng.

Với mô đun kết nối giữa client và server, đây là một thành phần rất quan trọng có tích chất quyết định đến kiến trúc mà hệ thống xây dựng. Một trong những yếu tố chính trong việc xây dựng mô đun kết nối là lựa chọn giao thức kết nối và truyền thông tin giữa client và server. Với một hệ thống game online trên di động em lựa chọn xây phương thức kết nối qua Socket TCP. Phần lớn các thiết bị di động hiện nay đều hỗ trợ kết nối qua Socket, hơn nữa kết nối qua Socket có độ tin cậy cao và tốc độ truyền dữ liệu khá tốt. Với vai trò rất quan trọng, mô đun kết nối cũng là thành phần cần được chú trọng xây dựng các phương án chịu lỗi để đảm bảo các thiết bị client luôn có thể kết nối tới server trong trường hợp có lỗi xảy ra.

Sau khi đã khái quát được mô hình và kiến trúc của hệ thống, một vấn đề khác cũng cần được lưu tâm để đảm bảo chất lượng của hệ thống là khả năng chịu lỗi. Các kĩ thuật phần mềm chịu lỗi đã được nghiên cứu từ rất lâu, rất nhiều các kĩ thuật đã được áp dụng và mang lại hiệu quả cao, vấn đề đặt ra là lựa chọn một kĩ thuật phù hợp nhất với hệ thống. Trong việc áp dụng các kĩ thuật chịu lỗi cũng cần đặc biệt chú ý tới các yếu tố về hiệu năng và thời gian phát triển hệ thống. Các kĩ thuật chịu lỗi thường đi kèm với các phép kiểm thử, các phương pháp ra quyết định, sự đa dạng dữ liệu, phòng ngừa lỗi, … vì vậy có thể gây giảm hiệu năng và tốc độ của hệ thống. Với những kĩ thuật chịu lỗi có yêu cầu các yếu tố dự phòng (dự phòng về phần cứng, dự phòng cơ sở dữ liệu, …) cũng sẽ gây ra tốn kém công sức và giá thành khi triển khai và bảo trì hệ thống. Hơn thế nữa việc phát triển các kĩ thuật chịu lỗi cũng gây ra sự phức tạp trong việc thiết kế và phát triển hệ thống. Vì vậy việc áp dụng các kĩ thuật chịu lỗi cần được xem xét thật kĩ để cân bằng tất cả các yếu tố, toàn bộ hệ thống không cần thiết phải hoàn toàn có khả năng chịu lỗi, chúng ta sẽ chỉ tập trung xây dựng mô đun chịu lỗi cho các thành phần thực sự quan trọng và dễ có nguy cơ xảy ra lỗi.

# CHƯƠNG 2. LÝ THUYẾT PHẦN MỀM CHỊU LỖI

## 2.1 Khái niệm chung về phần mềm chịu lỗi

### 2.1.1 Phần mềm chịu lỗi

Trong suốt 50 năm qua, máy tính đã phát triển từ những cỗ máy cồng kềnh và chỉ có trong các cơ quan của chính phủ hoặc các viện nghiên cứu lớn để trở thành những thiết bị hiện hữu mọi nơi trong cuộc sống của chúng ta. Chúng là các máy tính để bản, laptop, PDA, là thành phần rất quan trọng của xe hơi, đồ gia dụng, máy bay, các thiết bị y tế, …. Máy tính được sử dụng để vận hành các hệ thống sản xuất trong các nhà máy sản xuất, các nhà máy hạt nhân, các trạm vũ trụ, … Hệ thống máy tính cũng là nền tảng của hệ thống tài chính toàn cầu : Thực hiện những khối lượng giao dịch khổng lồ, phân tích tiền tệ, giao dịch trái phiếu và cổ phiếu, … Có thể thấy cuộc sống của chúng ta hiện tại ngày càng phụ thuộc nhiều vào máy tính.

Các hệ thống máy tính hiện tại ( bao gồm cả phần cứng và phần mềm ) ngày càng trở nên phức tạp và tinh vi hơn để đáp ứng nhu cầu của con người, kéo theo đó là việc luôn luôn tiềm ẩn nguy cơ xảy ra lỗi. Và một lỗi dù lớn hay nhỏ đều gây ra những ảnh hưởng và hậu quả nhất định. Các nhà khoa học và kĩ sư máy tính trong quá trình phát triển hệ thống vẫn luôn nghiên cứu và thiết kế ra hàng loạt các công cụ, mô hình và kĩ thuật để làm hạn chế lỗi trong hệ thống mà họ xây dựng. Tuy nhiên, như thế là chưa đủ: Chúng ta cần phải xây dựng hệ thống sẽ thừa nhận sự tồn tại của lỗi như là một thực tế của cuộc sống, và kết hợp các kỹ thuật để chịu đựng những lỗi lầm này trong khi vẫn cung cấp một mức độ chấp nhận được của dịch vụ. Các kết quả nghiên cứu được hình thành nên lý thuyết về chịu lỗi.

### 2.1.2 Phân loại lỗi

Trong ngôn ngữ hàng ngày, các từ “fault” (lỗi), “failure” (sai sót), “error” (thất bại) thường được dùng thay thế cho nhau. Nhưng trong lý thuyết về phần mềm chịu lỗi chúng có ý nghĩa đặc biệt. Một “fault” (hoặc “failure”) có thể là một khiếm khuyết của phần cứng hoặc cũng có thể là một sai lầm trong lập trình (hay còn gọi là “bug”). Ngược lại một “error” là một biểu hiện của “fault”, hoặc “failure”.

Ví dụ, hãy xem xét một bộ mạch cộng, với một đường dây đầu ra bị kẹt và luôn trả ra giá trị 1 mà không phụ thuộc vào giá trị của các toán hạng đầu vào. Đây là một “fault” nhưng chưa phải là một “error”. Những sai lầm này gây ra một thất bại trong việc thực hiện các phép toán có sử dụng đến bộ mạch cộng này. Một ví dụ tương tự là việc sai lầm trong lập trình và viết mã gây ra những lỗi khi thực thi chương trình. Chẳng hạn, có một chương trình con có nhiệm vụ tính toán hàm lượng giác sin(x) , nhưng do một sai lầm trong lập trình đoạn mã đó lại thực hiện việc tính giá trị tuyệt đối. Những sai lầm này sẽ dẫn tới lỗi khi thực hiện chương trình con với những giá trị đầu vào gây nhầm lẫn.

Cả “faults” và “errors” đều có thể lây lan qua hệ thống. Lỗi có thể lây lan bởi lẽ đầu ra của mô đun này sẽ trở thành đầu vào của các mô đun khác. Chằng hạn như ví dụ trên được xem xét, khi một bộ mạch cộng bị lỗi, kết quả sai sẽ khiến cho kết quả của toàn bộ phép tính logic bị sai từ đó gây thất bại cho một chương trình phần mềm. Để hạn chế sự lây lan như vậy xảy ra các nhà thiết kế thường tạo ra những “vùng chứa” trong hệ thống. Những “vùng chứa” này thực chất là các rào cản lỗi với nhiệm vụ hạn chế tối đa sự lây lan lỗi giữa các vùng khác nhau trong hệ thống. Một vùng chứa có thể được tạo ra bằng cách sử dụng các đơn vị hoặc chương trình dự phòng và thực hiện biểu quyết cho giá trị đầu ra của chúng.

Mối quan hệ của lỗi, sai sót và thất bại có thể được mô tả như một chuỗi (Laprie gọi nó là chuỗi cơ bản):

... failure fault error failure fault ...

Điều quan trọng là phải nhận thấy rằng không phải tất cả các lỗi đều dẫn đến thất bại, và không phải tất cả các thất bại đều dẫn đến sai sót. Lỗi được cho là hoạt động khi nó tạo ra một thất bại và không hoạt động trong trường hợp ngược lại. Một lỗi tiềm ẩn cho đến khi nó được công nhận như là một lỗi. Các lỗi được phát hiện bởi các thuật toán phát hiện lỗi hoặc các cơ chế phát hiện lỗi. Thất bại xảy ra khi một lỗi "đi qua" giao diện của hệ thống. Một lỗi có thể được xử lý đầy đủ bởi các quá trình trong đó các lỗi được phát hiện được xem như là một lỗi nội bộ. Một lỗi mà không thể có đủ xử lý bởi các quá trình trong đó nó được phát hiện, nhưng có tác dụng hạn chế quá trình đó là một lỗi bên ngoài. Một lỗi phổ biến là một lỗi mà không thể được xử lý đầy đủ bởi các quá trình trong đó nó được phát hiện và kết quả trong các lỗi trong các quá trình khác. Xem xét tỷ lệ mắc các lỗi có thể thực hiện một sự phân biệt giữa các lỗi liên tục, tức là lỗi xảy ra thường xuyên hơn so với một số ngưỡng được xác định trước, và các lỗi thoáng qua, lỗi tức là không liên tục. Trong thực tế, phân loại một lỗi chỉ có thể được xác định một cách tương đối vì nó sẽ là không thể để phân loại tất cả các lỗi một cách chính xác .

### 2.1.3 Lý thuyết độ tin cậy

**2.1.3.1 Độ tin cậy**

Độ tin cậy (dependability) được định nghĩa như là sự tin tưởng được đặt vào các dịch vụ mà hệ thống cung cấp. Để xem xét về độ tin cậy phần mềm chúng ta cùng nhau nhắc lại về định nghĩa của hệ thống và dịch vụ.

Hệ thống là một tập hợp các thành phần tương tác với nhau theo một thiết kế được quy định và kiểm soát bởi các mô hình tương tác. Hệ thống có thể tương tác với môi trường hoặc bị can thiệp bởi các thành phần của môi trường. Các ranh giới của hệ thống có thể khác nhau tùy thuộc vào cách nhìn ở những góc độ khác nhau. Ví dụ, người dùng có thể xem một hệ thống như một thực thể mà họ có thể tương tác trong khi các nhà thiết kế xem xét nó là một trong số các hệ thống tương tác với nhau (bao gồm cả người sử dụng). Các dịch vụ được hệ thống cung cấp là các hành vi của hệ thống được cảm nhận bởi “user”, một “user” là một hệ thống khác (con người hoặc không phải con người) tương tác với hệ thống.

Kể từ khi hệ thống máy tính được sử dụng trong nhiều lĩnh vực khác nhau có rất nhiều ứng dụng khác nhau mà tất cả có thể nhấn mạnh các khía cạnh khác nhau của độ tin cậy. Điều này có nghĩa là độ tin cậy có thể được xem theo nhiều tính chất khác nhau hay còn gọi là các thuộc tính. Chúng ta có bốn thuộc tính chính của độ tin cậy :

* Tính sẵn sàng: Mức độ mà một hệ thống có sẵn sàng để sử dụng.
* Độ tin cậy: Mức độ mà hệ thống liên tục cung cấp dịch vụ của mình.
* Độ an toàn: Mức độ mà một hệ thống tránh những hậu quả thảm khốc về môi trường.
* Độ bảo mật: Mức độ mà một hệ thống ngăn chặn truy cập trái phép.

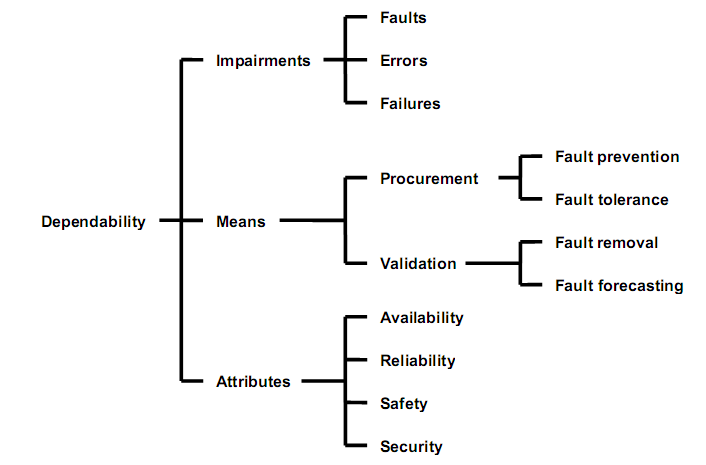
Bên cạnh đó có bốn nhóm các phương pháp nên được sử dụng trong quá trình phát triển các hệ thống máy tính đáng tin cậy :

* Phòng chống lỗi : Phòng chống sự xuất hiện của lỗi.
* Chịu lỗi: Hệ thống vẫn cung cấp các dịch vụ ở mức chấp nhận được khi xảy ra lỗi (Đã được trình bày kĩ trong phần 2.1).
* Loại bỏ lỗi: Giảm thiểu sự hiện diện của những lỗi trong hệ thống, cả về số lượng và mức độ nghiêm trọng của lỗi.
* Dự báo lỗi: Dự đoán và đánh giá mức độ nghiêm trọng của lỗi.

Hai phương pháp phòng chống lỗi và chịu lỗi có thể được coi như là những phương phát xây dựng hệ thống đáng tin cậy. Hai phương pháp loại bỏ lỗi và dự báo lỗi được dùng để thẩm định và đảm bảo độ tin cậy của hệ thống.

Qua đó, độ tin cậy phần mềm có thể được phân chia theo từng nhóm như sau :

* Những yếu tố làm suy yếu (Inpairments) : Những yếu tố đó là : “fault” (lỗi), “failure” (sai sót), “error” (thất bại). Những yếu tố này là nguyên nhân gây cho hệ thống mất đi tính tin cậy.
* Phương tiện (Means): Những phương tiện đó là: Phòng chống lỗi, Chịu lỗi, Loại bỏ lỗi, Dự báo lỗi.
* Những thuộc tính (Attributes): Giúp đánh giá và xác định độ tin cậy phần mềm.



Hình 1. Độ tin cậy

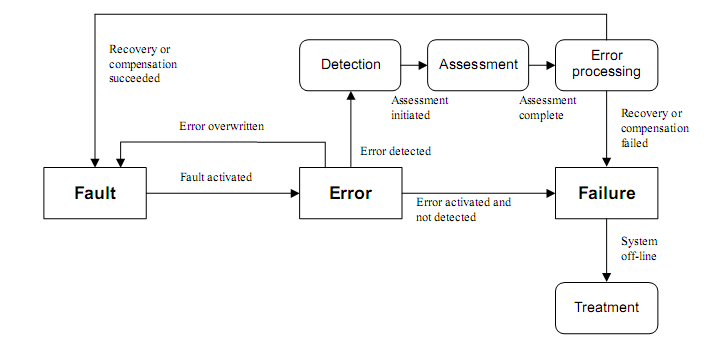
**2.1.3.2 Các phương thức của độ tin cậy**

Các phương pháp có thể được sử dụng để đạt được độ tin cậy là: Phòng chống lỗi, Chịu lỗi, Loại bỏ lỗi, Dự báo lỗi. Trong thực tế rất khó có thể áp dụng thành công cả bốn phương pháp với những mục tiêu của chúng. Chúng ta có thể phối hợp cả bốn phương pháp để tạo ra một hệ thống đáng tin cậy.

Cụm từ “phòng tránh lỗi” được sử dụng để mô tả sự liên kết chặt chẽ của loại bỏ lỗi và phòng ngừa lỗi. Phòng tránh lỗi là sử dụng các phương pháp để tạo ra một hệ thống không hề có lỗi. Ví dụ như bằng cách sử dụng các thành phần đáng tin cậy nhất, triển khai thực hiện các kỹ thuật tốt nhất cho các mối liên kết giữa các thành phần và thực hiện kiểm tra toàn diện để loại bỏ lỗi phần mềm và phần cứng.

Chịu lỗi là khả năng của một hệ thống hoạt động để chịu đựng sự hiện diện của lỗi, được thực hiện chủ yếu bằng quy trình xử lý lỗi và “điều trị lỗi”. Xử lý lỗi nhằm loại bỏ các lỗi trong trạng thái của hệ thống, điều trị lỗi nhằm mục đích ngăn ngừa các lỗi được kích hoạt lại. Để có thể thực hiện xử lý lỗi hệ thống phải phát hiện các lỗi và đánh giá thiệt hại gây ra bởi lỗi. Có bốn giai đoạn cần thực hiện để hệ thống có thể chịu lỗi :

* Phát hiện lỗi
* Đánh giá thiệt hại do lỗi gây ra cho hệ thống
* Xử lý lỗi (phục hồi hoặc đền bù)
* Điều trị lỗi (chẩn đoán và thụ động hóa)

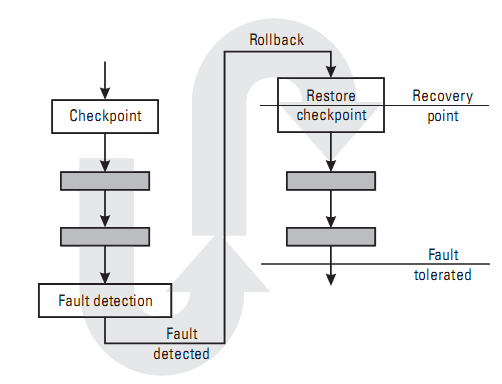


Hình 2. Bốn giai đoạn của chịu lỗi

Phát hiện lỗi là phát hiện một trạng thái có sai sót, tức là một trạng thái có khả năng dẫn tới một chuỗi các thất bại tiếp theo. Hiệu quả của các kỹ thuật để phát hiện lỗi là rất quan trọng cho sự thành công của bất kỳ hệ thống chịu lỗi nào.

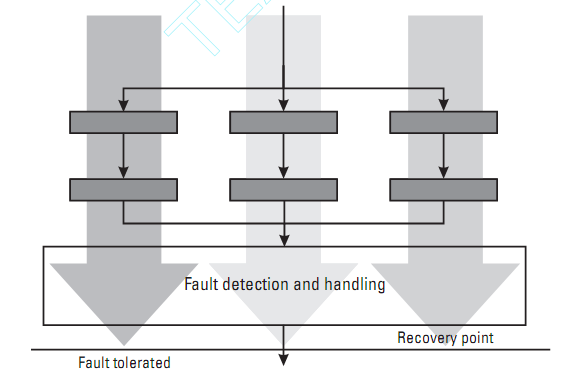
Đánh giá thiệt hại được thực hiện khi một lỗi đã được phát hiện để thiết lập chính xác hơn mức độ hệ thống bị hư hỏng. Những nhà thiết kế hệ thống sẽ dựa vào những đánh giá này để ngăn ngừa sự lan truyền của lỗi.

Khi thiệt hại cho hệ thống đã được đánh giá, lỗi có thể được xử lý. Điều này có thể được thực hiện theo hai cách: phục hồi lỗi hoặc bồi thường lỗi. Phục hồi là một nỗ lực để thay thế trạng thái lỗi hiện tại của hệ thống bởi một trạng thái khác không bị lỗi. Phục hồi có thể được thực hiện bằng “phục hồi về phía sau” (backward recovery) (H.3) hoặc “phục hồi về phía trước” (forward recovery) (H.4). Phục hồi về phía sau có nghĩa là hệ thống sẽ được đưa trở về một trạng thái không có lỗi được lưu lại tại một điểm khôi phục, một “ảnh chụp” của hệ thống trước khi có sai sót. Phục hồi về phía trước có nghĩa là sự chuyển đổi của trạng thái có sai sót bao gồm việc tìm kiếm và xây dựng một trạng thái mới mà từ đó hệ thống có thể hoạt động bình thường. Với bồi thường lỗi khi hệ thống ở trạng thái có lỗi vẫn phải chứa đầy đủ các phương án dự phòng để cung cấp đầy đủ các dịch vụ không có lỗi từ trạng thái bị lỗi.



Hình 3. Phục hồi lỗi về phía sau (Backward Recovery)

Khi hệ thống đã chịu lỗi thành công, hệ thống có thể tiến hành điều trị lỗi. Bước đầu tiên trong điều trị lỗi là chẩn đoán lỗi, trong đó bao gồm xác định nguyên nhân của các lỗi, đối với cả vị trí và tính chất. Các biện pháp này hiếm khi được sử dụng với các hệ thống gặp phải lỗi mà vẫn đang hoạt động. Nếu biện pháp này được thực hiện đầy đủ có thể ngăn chặn lỗi được kích hoạt một lần nữa.



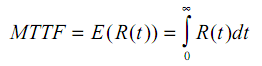
Hình 4. Phục hồi về phía trước (Forward Recovery)

Một bước quan trọng trong việc xây dựng các hệ thống đáng tin cậy là loại bỏ lỗi, trong đó bao gồm ba bước: xác minh, chẩn đoán và sửa lỗi.Xác minh là quá trình kiểm tra xem hệ thống có tuân thủ các thuộc tính nhất định hay không, vì vậy được gọi là điều kiện xác minh.Nếu không tuân thủ, hai bước khác phải được thực hiện: chẩn đoán các lỗi ngăn cản các điều kiện xác minh được thực hiện, và sau đó thực hiện các sửa chữa cần thiết. Sau khi điều chỉnh, các quá trình xác minh được bắt đầu từ đầu.

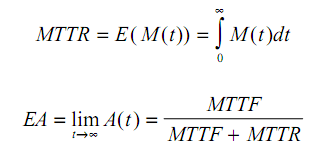
**2.1.3.3 Các thuộc tính của độ tin cậy**

Các thuộc tính của độ tin cậy cho phép một hệ thống phù hợp với mục tiêu đáng tin cậy được thể hiện.Những thuộc tính này sau đó có thể sử dụng như là độ đo mức độ tin cậy của hệ thống và có thể được nhấn mạnh nhiều hơn hoặc ít hơn, tùy thuộc vào ứng dụng dành cho hệ thống máy tính đang xem xét. Các thuộc tính đáng tin cậy chính là:

* Độ tin cậy (Reliability) : Thuộc tính này có thể được đặc trưng bởi một hàm R (t) thể hiện khả năng rằng một hệ thống phù hợp với thông số kỹ thuật của nó trong suốt một khoảng thời gian t. Tính tin cậy của một hệ thống phải có tỉ lệ nghịch với tốc độ mà thất bại xảy ra. Độ đo tương ứng được sử dụng là Mean Time To Failure (MTTF) :



* Tính sẵn sàng (Availability) : Thuộc tính này có thể được đặc trưng bởi một hàm A (t), trong đó thể hiện khả năng mà hệ thống sẽ phù hợp với đặc điểm kỹ thuật của nó tại một thời điểm t. Độ đo tương ứng là EA (Expected Availability). Tuy nhiên với định nghĩa của EA chúng ta cũng cần quan tâm tới độ đo MTTR. MTTR được đặc trưng bởi một hàm M (t) thể hiện khả năng mà một hệ thống sẽ được sửa chữa trước khi thời điểm t.



* Độ an toàn : Thuộc tính này được đặc trưng bởi hàm S(t). Thể hiện khả năng hệ thống sẽ vẫn an toàn trong suốt một thời gian t. Độ đo tương ứng là MTTC (Mean Time To Catastrophic failure).
* Độ bảo mật : Mức độ mà một hệ thống ngăn chặn truy cập trái phép.

### 2.1.4 Lý thuyết dự phòng (Redundancy)

Một trong những khía cạnh rất quan trọng của hệ thống chịu lỗi là khai thác và quản lý dự phòng. Dự phòng là hệ thống có một lượng tài nguyên nhiều hơn lượng tài nguyên tối thiểu cần thiết để thực hiện các dịch vụ. Khi lỗi xảy ra, các phương án dự phòng được sử dụng để che giấu hoặc bao trùm lên các lỗi và vẫn duy trì mức độ mong muốn của các chức năng.

Trong lý thuyết về dự phòng có bốn hình thức chính được áp dụng : Dự phòng phần cứng, phần mềm, thông tin và thời gian. Lỗi phần cứng thường được xử lý bằng cách sử dụng dự phòng phần cứng, thông tin, hoặc thời gian, trong khi lỗi phần mềm được xử lý bằng cách dự phòng phần mềm.

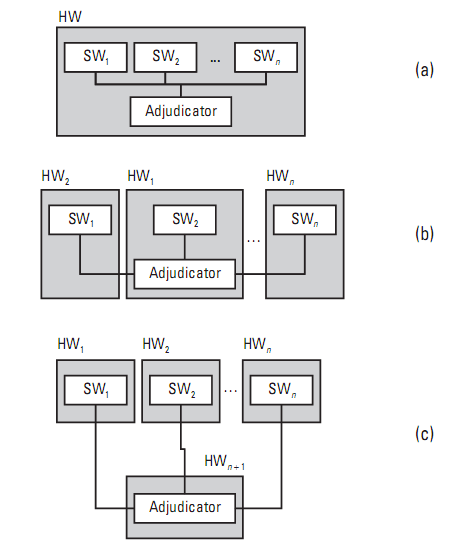
Dự phòng phần cứng được thực hiện bằng cách trong thiết kế chúng ta thêm vào các thành phần mở rộng để phát hiện và ghi đè lên các tác động của một thành phần gặp lỗi. Ví dụ, thay vì có một bộ xử lý duy nhất, chúng ta có thể sử dụng hai hoặc ba bộ xử lý, thực hiện chức năng tương tự. Khi có hai bộ xử lý chúng ta có thể phát hiện ra lỗi của bộ vi xử lý đơn. Khi có ba bộ vi xử lý chúng ta có thể xử dụng hai bộ vi xử lý để ghi đè lên kết quả đầu ra của bộ vi xử lý gặp lỗi. Đây là một ví dụ của dự phòng phần cứng tĩnh, mục tiêu chính trong đó là bo trùm ngay lập tức thất bại. Một hình thức khác là dự phòng phần cứng động, khi các thành phần dự phòng sẽ được kích hoạt khi có lỗi xảy ra với thành phần đang hoạt động. Một sự kết hợp giữa dự phòng phần cứng tĩnh và động là dự phòng phần cứng lai.

Những biện pháp dự phòng phần cứng phát sinh chi phí khá cao, vì vậy nó chủ yếu được sử dụng cho các hệ thống lớn và đặc biệt quan trọng nới các lỗi phần cứng khi xảy ra có thể gây ra những hậu quả rất nghiêm trọng.

Các phương pháp hay được sử dụng nhất của dự phòng thông tin là phát hiện lỗi và mã sửa chữa. Ở đây, bit mở rộng (được gọi là bit kiểm tra) được thêm vào các bit dữ liệu ban đầu, như vậy là một lỗi trong các bit dữ liệu có thể được phát hiện hoặc thậm chí sửa chữa ngay. Kết quả phát hiện lỗi và mã sửa lỗi hiện nay được sử dụng rộng rãi trong các đơn vị bộ nhớ và các thiết bị lưu trữ khác nhau để bảo vệ thông tin chống lại thất bại. Lưu ý rằng các mã lỗi (giống như bất kỳ hình thức khác của sự dư thừa thông tin) yêu cầu thêm phần cứng để xử lý các dữ liệu dư thừa (các bit kiểm tra).

Dự phòng thời gian có hiệu quả chủ yếu chống lại các lỗi thoáng qua. Bởi vì phần lớn các lỗi phần cứng là thoáng qua, nó không chắc chắn rằng các hành động thực hiện riêng biệt sẽ trải nghiệm những lỗi tương tự. Dự phòng thời gian do đó có thể được sử dụng để phát hiện lỗi thoáng qua trong các tình huống trong đó những lỗi lầm như vậy có thể bị phát hiện hoặc không bị phát hiện. Dự phòng thời gian cũng có thể được sử dụng cùng các phương tiện khác để phát hiện lỗi trong hệ thống có khả năng phục hồi từ những ảnh hưởng của lỗi và lặp đi lặp lại việc tính toán. So với các hình thức khác của sự dự phòng, dự phòng thời gian có tổng chi phí phần cứng và phần mềm thấp hơn nhiều.

Dự phòng phần mềm được sử dụng chủ yếu để chống lại thất bại phần mềm.Nó là một dự đoán hợp lý rằng tất cả các module lớn của phần mềm đã từng được sản xuất đều có lỗi. Đối phó với những lỗi lầm như vậy có thể rất tốn kém, một cách thường được áp dụng là tiến hành độc lập sản xuất hai hoặc nhiều phiên bản của phần mềm đó (tốt hơn là bởi các đội lập trình khác nhau) với hy vọng rằng các phiên bản khác nhau sẽ không thất bại trên cùng một đầu vào. Phiên bản thứ hai (hoặc các phiên bản khác) có thể được dựa trên các thuật toán đơn giản và ít chính xác hơn (và, do đó, ít có khả năng có lỗi) được sử dụng chỉ khi sự thất bại của phần mềm chính để sản xuất các kết quả chấp nhận được. Cũng như đối với dự phòng phần cứng, nhiều phiên bản của chương trình có thể được thực hiện hoặc đồng thời (yêu cầu phần cứng dự phòng) hoặc tuần tự (yêu cầu thêm thời gian, tức là, thời gian dự phòng) khi phát hiện sự thất bại.



Hình 5. Dự phòng phần mềm

## 2.2 Các kĩ thuật xây dựng phần mềm chịu lỗi

### 2.2.1 Kĩ thuật phần mềm chịu lỗi với một phiên bản phần mềm

Kĩ thuật phần mềm chịu lỗi cho một phiên bản phần mềm được dựa trên việc sử dụng các phương pháp dự phòng được áp dụng cho một phiên bản duy nhất của phần mềm để phát hiện và phục hồi lỗi. Các kĩ thuật được áp dụng bao gồm : Cân nhắc về cấu trúc và hành động của chương trình, phát hiện lỗi, xử lý ngoại lệ, checkpoint và khởi động lại, đa dạng dữ liệu.

**2.2.1.1 Phát hiện lỗi**

Khi áp dụng kĩ thuật chịu lỗi trong một phiên bản phần mềm, để tăng cường hiệu quả các mô đun thường phải có hai thuộc tính cơ bản : Tự bảo vệ và tự kiểm tra. Thuộc tính tự bảo vệ có nghĩa là một thành phần phải có khả năng tự bảo vệ mình khỏi bị “lây nhiễm lỗi” từ bên ngoài bằng cách phát hiện sai sót trong thông tin được truyền cho mô đun từ các thành phần khác trong quá trình tương tác với mô đun. Tự kiểm tra có nghĩa là một thành phần phải có khả năng phát hiện những lỗi nội bộ và có những hành động thích hợp để ngăn chặn sự lan truyền của những sai sót đến các thành phần khác. Lưu ý rằng dự phòng chịu lỗi không có đóng góp nhiều cho chức năng hệ thống, nhưng thay vào đó là chất lượng của sản phẩm. Tương tự như vậy, cơ chế phát hiện lỗi làm giảm hiệu năng hệ thống. Thực tế sử dụng khả năng chịu lỗi trong thiết kế được dựa trên sự cân đối với các yếu tố chức năng, hiệu suất, phức tạp, và an toàn.

Phát hiện lỗi có thể được phân chia thành các kĩ thuật khác nhau, các kĩ thuật này có thể được áp dụng để đảm bảo các thuộc tính của mô đun như đã đề cập ở phần trên :

* Kiểm tra nhân rộng được thực hiện bằng cách đối chiếu các thành phần với phát hiện lỗi dựa trên so sánh các kết quả đầu ra.
* Kiểm tra thời gian được áp dụng cho các hệ thống và các mô đun có các thông số kĩ thuật bao gồm cả hạn chế thời gian. Dựa trên những hạn chế này kiểm tra có thể được phát triển để tìm độ lệch từ các hành vi có thể chấp nhận được của mô đun.
* Kiểm tra ngược sử dụng đầu ra của mô đun để tính toán đầu vào dựa trên những chức năng của mô đun. Lỗi được phát hiện khi các yếu tố đầu vào tính toán được không phù hợp với các yếu tố đầu vào thực tế. Kiểm tra ngược được áp dụng với những mô đun mà việc tính toán đầu ra là tương đối đơn giản.
* Kiểm tra mã sử dụng dự phòng trong các đại diện của các thông tin với mối quan hệ giữa thông tin thực tế và thông tin dưa thừa. Phát hiện lỗi được dựa trên việc kiểm tra trước và sau khi các phép tính được thực hiện.
* Kiểm tra tính hợp lý được sử dụng để đánh giá thuộc tính ngữ nghĩa của dữ liệu (ví dụ như phạm vi, tốc độ thay đổi và trình tự) từ đó phát hiện lỗi.
* Kiểm tra cấu trúc sử dụng những thuộc tính của cấu trúc dữ liệu. Ví dụ danh sách, queue, stack, cây, … có thể được kiểm tra số yếu tố trong cấu trúc, liên kết và con trỏ của chúng, và bất kỳ thông tin cụ thể mà có thể là khớp nối.

Cây lỗi đã được đề xuất như là một trợ giúp thiết kế trong việc phát triển các chiến lược phát hiện lỗi. Cây lỗi có thể được sử dụng để xác định các lớp của thất bại và điều kiện có thể gây ra những thất bại. Cây lỗi đại diện cho một cách tiếp cận top-down, mặc dù không đảm bảo có thể xác định đầy đủ, nhưng nó rất hữu ích trong các tài liệu giả định, đơn giản hóa các đánh giá của thiết kế, xác định thiếu sót, và cho phép các nhà thiết kế để hình dung tương tác giữa các thành phần và hậu quả của họ thông qua các phương tiện có cấu trúc đồ họa. Cây lỗi cho phép các nhà thiết kế để thực hiện phân tích chất lượng của sự phức tạp và mức độ độc lập trong kiểm tra lỗi của một chiến lược đề xuất khả năng chịu lỗi. Nói chung, khi một cây lỗi được xây dựng, cấu trúc của cây đi từ khái niệm chức năng cao cấp các yếu tố thiết kế phụ thuộc nhiều hơn.

**2.2.1.2 Xử lý ngoại lệ**

Xử lý ngoại lệ là thực hiện việc gián đoạn các chức năng thông thường để xử lý các chức năng bất thường. Trong kỹ thuật chịu lỗi phần mềm, ngoại lệ được báo hiệu bằng việc thực hiện các cơ chế phát hiện lỗi. Thiết kế xử lý ngoại lệ đòi hỏi phải xem xét các sự kiện có thể kích hoạt trường hợp xử lý ngoại lệ, ảnh hưởng của những sự kiện đến hệ thống và lựa chọn các hành động giảm nhẹ thích hợp. Các sự kiện kích hoạt ngoại lệ được chia ra thành 3 lớp : Giao diện ngoại lệ, ngoại lệ nội bộ và các ngoại lệ thất bại.

* Giao diện ngoại lệ được báo hiệu bởi một thành phần khi nó phát hiện một yêu cầu dịch vụ không hợp lệ. Đây là loại ngoại lệ được kích hoạt bởi cơ chế tự bảo vệ của một module khi gặp những ngoại lệ.
* Ngoại lệ nội bộ được báo hiệu bởi một mô-đun khi cơ chế phát hiện lỗi của nó tìm thấy một lỗi trong hoạt động nội bộ của mình.Những trường hợp ngoại lệ này nên được xử lý bởi khả năng chịu lỗi của mô-đun.
* Trường hợp ngoại lệ thất bại được báo hiệu bởi một mô-đun sau khi nó đã phát hiện một lỗi mà cơ chế xử lý lỗi của nó đã không thể xử lý thành công.Trong thực tế, trường hợp ngoại lệ thất bại cho các module yêu cầu dịch vụ mà một số phương tiện khác phải được tìm thấy để thực hiện chức năng của nó.

Nếu cấu trúc hệ thống và cơ chế phát hiện lỗi thiết kế đúng cách, những ảnh hưởng của lỗi sẽ được chứa trong một tập hợp các thành phần tương tác tại thời điểm lỗi được phát hiện. Kiến thức về ngăn chặn lỗi này là cần thiết để thiết kế xử lý ngoại lệ có hiệu quả.

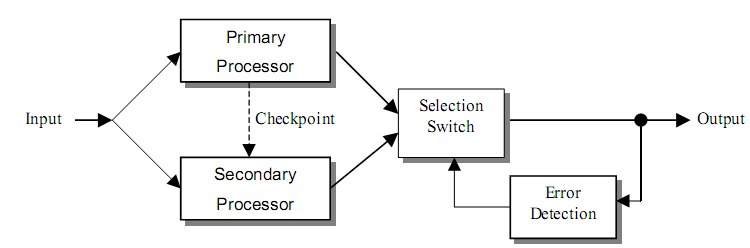
**2.2.1.3 Checkpoint và Restart**

Với kĩ thuật chịu lỗi cho một phiên bản phần mềm có một số phương pháp để thực hiện phục hồi. Thường được nhắc đến nhiều nhất là phương pháp “điểm phục hồi và khởi động lại”. Với kĩ thuật này chỉ cần khởi động lại một mô-đun thường là đủ để cho phép hoàn thành nhiệm vụ của hệ thống trước khi bị lỗi.Một khởi động lại, hoặc phục hồi lỗi về phía sau, có ưu điểm là độc lập với thiệt hại gây ra bởi một lỗi, áp dụng đối với lỗi không lường trước được, nói chung cơ chế này có thể được sử dụng ở nhiều cấp độ trong một hệ thống, và khái niệm cũng khá đơn giản.

Có hai loại phục hồi khởi động lại : phục hồi khởi động lại tĩnh và động. Khởi động lại dạng tĩnh dựa trên việc trở lại một trạng thái được xác định trước. Nó có thể trực tiếp trở lại một trạng thái đã được khởi tạo từ lúc đầu, hoặc là một trong một tập hợp các trạng thái có thể, với sự lựa chọn được thực hiện dựa trên tình hình hoạt động tại thời điểm phát hiện lỗi xảy ra. Khởi động lại động sử dụng các điểm kiểm tra tự động (checkpoint) tạo ra các bức ảnh chụp của trạng thái hệ thống tại các điểm khác nhau trong quá trình thực hiện. Các checkpoint có thể được tạo ra trong khoảng thời gian cố định hoặc tại các điểm cụ thể trong quá trình thực hiện và được xác định bởi một số nguyên tắc tối ưu hóa. Lợi thế của checkpoint là dựa trên các trạng thái được tạo ra trong quá trình hoạt động, và do đó có thể được sử dụng để cho phép tiến trình chuyển tiếp thực hiện mà không cần phải loại bỏ tất cả các công việc thực hiện lên đến thời điểm phát hiện lỗi.

**2.2.1.4 Phương pháp cặp xử lý**

Một cặp quá trình xử lý sử dụng hai phiên bản giống nhau của phần mềm và chạy trên những bộ xử lý riêng biệt. Cơ chế phục hồi là checkpoint và restart. Chúng ta gọi hai bộ xử lý này là bộ xử lý chính và bộ xử lý thứ cấp. Lúc đầu hoạt động bộ xử lý chính xử lý thông tin đầu vào và đưa ra kết quả đầu ra, trong khi đó bộ nhớ thứ cấp sẽ tạo ra các checkpoint. Khi lỗi xảy ra bộ xử lý thứ cấp chạy lại các checkpoint và coi đó là trạng thái bắt đầu của nó và làm mất vai trò của bộ xử lý chính. Khi điều này xảy ra, bộ xử lý gặp lỗi sẽ ở trạng thái “offline” và tiến hành kiểm tra chẩn đoán lỗi. Sau khi trở lại phục vụ, bộ xử lý gặp lỗi sẽ trở thành bộ xử lý thứ cấp và thực hiện tạo các checkpoint từ bộ xử lý chính. Ưu điểm chính của kỹ thuật này là việc cung cấp các dịch vụ thực hiện liên tục không bị gián đoạn sau khi sự xuất hiện của một sự thất bại trong hệ thống.



Hình 6. Cặp xử lý

### 2.2.2 Kĩ thuật phần mềm chịu lỗi với nhiều phiên bản phần mềm

Khả năng chịu lỗi phần mềm cho nhiều phiên bản được dựa trên việc sử dụng hai hoặc nhiều phiên bản (hoặc "biến thể") của phần mềm, các phiên bản này được thực hiện tuần tự hoặc song song. Các phiên bản được sử dụng như là sự lựa chọn thay thế (với một phương tiện riêng biệt phát hiện lỗi) trong một cặp hoặc các nhóm lớn. Lý do cho việc sử dụng nhiều phiên bản là hi vọng rằng các thành phần được xây dựng khác nhau (tức là có nhà thiết kế khác nhau, các thuật toán khác nhau, các công cụ thiết kế khác nhau, …) sẽ có những sai phạm khác nhau. Vì vậy, nếu một phiên bản không thành công trên một đầu vào cụ thể, thì hi vọng rằng có ít nhất một trong các phiên bản thay thế có thể cung cấp một đầu ra thích hợp.

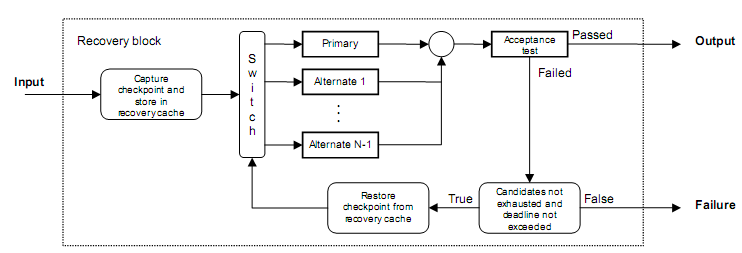
**2.2.2.1 Khối phục hồi (Recovery Block)**

**2.2.2.1.1 Lý thuyết cơ bản**

Khối phục hồi (Recovery Block – RB) được hiểu như là một lý thuyết về dự phòng phần mềm trong đó hệ thống có một kênh phần cứng, tức là không có dự phòng trong phần cứng, có dự phòng trong phần mềm - sử dụng nhiều mô-đun khác nhau thực hiện các nhiệm vụ tương tự nhau, và cũng là dư thừa trong thời gian, được tính khi nhiều mô-đun được thực hiện tuần tự trong trường hợp lỗi. Những yếu tố cơ bản của RB là :

* Một mô đun chính : Mô đun chương trình thực hiện các chức năng mong muốn (đây chính là chương trình thông thường).
* Không hoặc nhiều các mô đun thay thế : Các mô đun này có thể thực hiện các chức năng mong muốn của mô đun chính nhưng theo một cách khác.
* Bộ kiểm thử chấp nhận : Bộ kiểm thử thực hiện kiểm tra kết quả đầu ra của mô đun chính và các mô đun thay thế để xác nhận rằng kết quả thực hiện là chấp nhận được với môi trường của RB.

Để RB có thể cung cấp khả năng chịu lỗi và tiếp tục dịch vụ trong trường hợp mô đun chính gặp thất bại bởi lỗi, phải có ít nhất một mô đun thay thế. Các mô đun thay thế nên có sự khác biệt trong xử lý so với mô đun chính và khác biệt cả với những mô đun thay thế khác. Nếu các mô đun có những phương thức xử lý giống nhau thì sẽ gặp những thất bại giống nhau trên cùng một bộ dữ liệu đầu vào khi đó điều kiện của chịu lỗi sẽ không được đáp ứng. Các mô đun của khối phục hồi có thể kết hợp với các khối phục hồi bên trong, tức là chương trình phục hồi có thể lồng nhau. Bộ kiểm thử chấp nhận sử dụng để ra quyết định rằng kết quả được thực hiện bởi một mô đun có thể chấp nhận được hay không. Ngoài ra khối phục hồi còn yêu cầu một bộ nhớ “cache” phục hồi, tức là một cấu trúc cung cấp chức năng lưu trữ thông tin thiết yếu của trạng thái hệ thống ở thời điểm hiện tại (tạo ra các “checkpoint”).



Hình 7. Kiến trúc cơ bản của khối phục hồi

Khi dữ liệu được đưa vào khối phục hồi, một “checkpoint” được thành lập. Trạm kiểm soát này được lưu trữ trong bộ nhớ phục hồi đệm và chứa tất cả các dữ liệu liên quan mô tả trạng thái hiện tại của hệ thống được nhìn thấy từ khối phục hồi, tức là chỉ có các bộ phận của hệ thống dữ liệu có liên quan cho các khối phục hồi cần phải được lưu trữ. Sau khi “checkpoint” đã được lập, mô đun chính bắt đầu hoạt động. Sau khi thực hiện kết quả đầu ra được đưa vào bộ kiểm thử chấp nhận. Nếu kết quả vượt qua được bộ kiểm thử chấp nhận, đó sẽ là kết quả cuối cùng được đưa ra của khối phục hồi. Còn trường hợp khác là kết quả không được vượt qua. Có bốn nguyên nhân chính khiến bộ kiểm thử từ chối kết quả của mô đun :

* Một lỗi trong hoạt động “rõ ràng” của mô đun và bị bộ kiểm thử phát hiện ra.
* Việc ngắt mô đun gặp lỗi, được phát hiện bởi thời gian giới hạn “time out”.
* Lỗi tiềm ẩn trong mô đun (ví dụ như chia cho số 0).
* Một khối phục hồi bên trong khối phục hồi hiện tại có xảy ra lỗi.

Nếu kết quả không được chấp nhận, một thủ tục phục hồi được tiến hành. Thủ tục này sẽ khôi phục lại trạng thái hệ thống đã được sao lưu trước đó tại “checkpoint”. Khi phục hồi được hoàn thành về trạng thái ban đầu, mô đun thay thế đầu tiên được thực thi và đầu ra một lần nữa được đưa vào bộ kiểm thử chấp nhận. Nếu kiểm thử thành công, khối phục hồi sẽ hoàn thành, nếu không trạng thái hệ thống một lần nữa được phục hồi theo cách tương tự như trước và mô đun thay thế thứ hai được thực hiện. Công việc sẽ được hoàn thành khi xảy ra một trong hai trường hợp :

* Một mô đun thực hiện được kết quả đầu ra vượt qua được bộ kiểm thử chấp nhận.
* Tất cả các mô đun đều không thành công và khối phục hồi sẽ trả ra kết quả lỗi tới môi trường.

Một trường hợp đặc biệt của khối phục hồi là chỉ có một mô đun chính duy nhất và không hề có mô đun thay thế. Khi đó khối phục hồi sẽ không cần “checkpoint”, bộ nhớ đệm để lưu trạng thái hệ thống. Khi mô đun chính xảy ra lỗi, kết quả sẽ được trả ra ngay lập tức.

**2.2.2.1.2 Thiết kế khối phục hồi**

**a. Thiết kế mô đun chính và mô đun thay thế**

Một điểm quan trọng của khối phục hồi là cả mô đun chính và mô đun thay thế đều bắt đầu hoạt động ở cùng một trạng thái của hệ thống. Điều này giúp việc thiết kế các mô đun được diễn ra độc lập, mô đun này được thiết kế mà không cần biết đến cách thức hoạt động của mô đun khác. Các nhà thiết kế chương trình chứa khối phục hồi cũng không cần quan tâm tới mô đun sẽ đưa ra kết quả cuối cùng của khối phục hồi.

Ở đây chúng ta nhắc lại khái niệm về sự đa dạng phần mềm. Thực tế đã chứng minh rằng hoạt động độc lập của một thành phần trong hệ thống là không đa dạng, tức là chúng sẽ không thất bại một cách độc lập. Điều này chỉ ra rằng mặc dù các đội của các lập trình viên là khác nhau về kĩ thuật, trình độ, … thì các khuyết tật trong quá trình phát triển mà sau này xuất hiện như là lỗi trong chương trình cũng không hề độc lập.

Nếu mô đun chính được thiết kế để đảm bảo toàn bộ các dịch vụ cần thiết của hệ thống và các mô đun thay thế lại được thiết kế với sự đáp ứng các dịch vụ ngày càng giảm sẽ giúp cho việc phát triển các mô đun thay thế ít gặp lỗi hơn. Khi đó việc thiết kế các mô đun thay thế sẽ trở nên đơn giản hơn.

**b. Thiết kế kiểm thử chấp nhận**

Kiểm thử chấp nhận của một khối phục hồi có thể được coi là một sự khẳng định về tác động của việc thực hiện của một khối phục hồi, điều đó cần thiết cho việc đảm bảo thực hiện các hoạt động chính xác của chương trình xung quanh khối phục hổi. Kiểm thử cung cấp một phương pháp ra quyết định để xem xét liệu các kết quả đã được chấp nhận hay chưa, qua đó chỉ ra liệu các kết quả của mô đun có thể được chấp nhận bởi các chương trình xung quanh. Mỗi khối phục hồi chỉ có duy nhất một bộ kiểm thử cho tất cả các kết quả đầu ra của các mô đun trong khối phục hồi. Kiểm thử chấp nhận thường rơi vào một trong số các trường hợp sau : (a) Đáp ứng đủ yêu cầu, (b) Kiểm tra đánh giá, (c) Kiểm tra tính hợp lý và (d) Kiểm tra thời gian chạy của máy tính. Sự khác biệt giữa chúng đôi khi sẽ bị xóa mờ.

Lý tưởng nhất, một bộ kiểm thử chấp nhận có thể chỉ ra chắc chắn rằng kết quả là đúng hay sai. Tuy nhiên thực tế điều này rất ít khi xảy ra vì những lí do sau :

* Hiệu năng của bộ kiểm thử ở mức thấp.
* Các mô đun thay thế có thể không thực hiện đầy đủ các chức năng như mô đun chính.
* Sự phức tạp trong thiết kế một bộ kiểm thử khiến bản thân bộ kiểm thử cũng có khả năng xảy ra lỗi.

Với những vấn đề chỉ ra ở trên chúng ta cũng có thể nhận thấy rằng bộ kiểm thử cũng có thể xảy ra lỗi thiết kế phần mềm khi chính các kiểm tra của nó cũng được thực thi bằng phần mềm. Các phép toàn mà bộ kiểm thử cần thực hiện sẽ phức tạp hơn rất nhiều so với các phép toàn được các mô đun thực hiện. Để phát hiện được hết các lỗi khi tiến hành kiểm thử một mô đun, bộ kiểm thử cần phải biết đầy đủ về các đặc tính thực hiện của mô đun đó. Chính điều này khiến kiểm thử chấp nhận có khả năng cao bị xảy ra lỗi. Bộ kiểm thử chấp nhận là một yếu tố rất quan trọng trong việc xác định độ tin cậy của hệ thống. Một kiểm thử chấp nhận không đúng sẽ làm giảm độ tin cậy của hệ thống bất chấp việc hệ thống có mô đun chính với độ tin cậy rất cao.

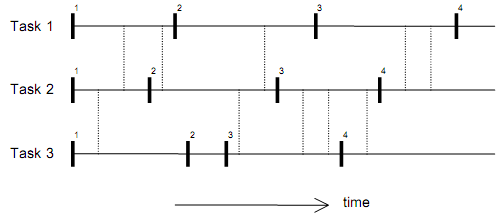
**c. Thiết kế cơ chế bộ nhớ đệm phục hồi**

Thông thường các nhà lập trình cho rằng để khôi phục lại trạng thái của hệ thống thì chỉ cần lưu trữ các biến toàn cục vì hầu hết các hoạt động trong các mô đun đều sử dụng biến địa phương. Điều này khiến việc thiết kế bộ nhớ đệm phục hồi trở nên khá đơn giản. Tuy nhiên trong một số trường hợp có thể có một số đối tượng có thể không hoặc không nên được đặt trong bộ nhớ đệm phục hồi. Để có thể khôi phục lại trạng thái của hệ thống trong trường hợp phục hồi, các mô-đun không được truy cập trực tiếp vào hệ thống xung quanh đang vận hành nó. Vấn đề này có thể được giải quyết bằng cách sử dụng hệ thống đa cấp, tức là hệ thống được xây dựng với nhiều cấp độ trừu tượng, và theo cách đó cung cấp cho đối tượng phục hồi được xây dựng trên các đối tượng không thể phục hồi.

Bộ nhớ đệm phục hồi được coi như là cốt lõi của khối phục hồi và nó luôn được giả định rằng là đáng tin cậy và không bao giờ thất bại. Các nhà thiết kế lập luận rằng thiết kế của một bộ nhớ đệm phục hồi là đủ đơn giản để nó có thể được đảm bảo rằng không có lỗi xảy ra.

**2.2.2.1.3 Hiệu ứng Domino**

Ban đầu, khối phục hồi chỉ xem xét đến các chương trình đơn tiến trình với cấu trúc tuần tự. Nhưng trong nhiều ứng dụn hiện tại (đặc biệt là các phần mềm nhúng) chương trình thực hiện đồng thời rất nhiều tác vụ có sự giao tiếp với nhau. Ta hãy xem xét ví dụ dưới đây khi 3 quá trình sử dụng các khối phục hồi và tương tác với nhau. Mỗi thanh dọc quy định một điểm phục hồi, các thanh nét đứt mô tả tương tác giữa các tác vụ.

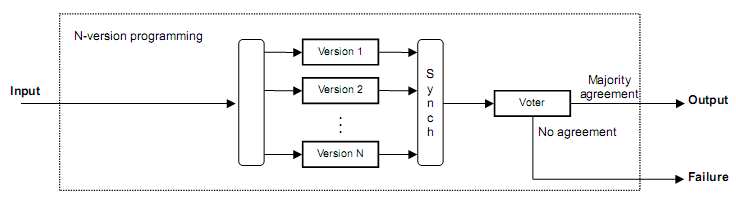


Hình 8. Hiệu ứng Domino

Nếu tác vụ 1 thất bại nó sẽ sao lưu điểm phục hồi cuối cùng của nó, tức là điểm phục hồi 4. Hai tác vụ khác không hề bị ảnh hưởng. Tương tự nếu tác vụ 2 thất bại nó cũng sẽ được triển khai trở lại từ điểm phục hồi thứ 4 của mình. Khi đó nó sẽ tương tác với tác vụ 1, tác vụ 1 cần thiết phải quay trở lại điểm khôi phục trước khi xảy ra tương tác tức là điểm phục hồi 3. Nếu tác vụ 3 thất bại tất cả các tác vụ sẽ phải quay trở lại điểm phục hồi đầu tiên. Hiệu ứng quay lại điểm phục hồi không thể kiểm soát được như vậy gọi là hiệu ứng Domino.

**2.2.2.2 N-Version Programming (NVP)**

Khái niệm của việc sử dụng nhiều phép tính toán để phát hiện và sửa chữa thất bại trong những phép tính toán đã được biết đến kể từ khi Babbage xây dựng công cụ tính toán của mình. Hiện tại, việc sử dụng nhiều phiên bản của một mô đun phần mềm để tha thứ những lỗi lầm đã được sử dụng từ những năm 1960, nhưng những nỗ lực điều tra các đặc điểm và hạn chế của phương pháp này đã không thực hiện cho đến khi Avizienis giới thiệu một khái quát của phương pháp sử dụng nhiều phép tính toán gọi là “N phiên bản lập trình” (N-Version Programming, NVP) vào năm 1977. Ông định nghĩa rằng NVP như là các thế hệ độc lập của N (với N ≥ 2) chương trình tương đương nhau về chức năng và được thiết kế từ những đặc điểm kĩ thuật ban đầu. N phiên bản này sẽ chạy trên một vài kênh phần cứng khác nhau và trả ra kết quả thông qua một cơ chế quyết định, được coi là một “cử tri” (voter). Nếu đa số các phiên bản trong N phiên bản này đồng ý với kết quả, kết quả này sẽ được coi như kết quả đầu ra. Nếu không có sự chấp nhận bởi đa số, hệ thống được coi là thất bại. Mô hình phía dưới cho chúng ta biết về cấu trúc của một mô hình NVP.



Hình 9. Cấu trúc N-Version Programming

Các yếu tố cơ bản cho việc tiếp cận NVP là :

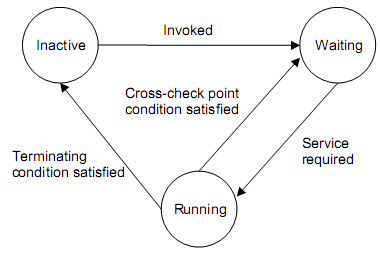
* Các đặc điểm kĩ thuật cơ bản : đây là đặc điểm kỹ thuật của các chức năng được mong muốn bởi phần mềm.
* N phiên bản phần mềm : Các mô đun phần mềm được tạo ra từ các đặc điểm kĩ thuật ban đầu.
* Một cơ chế ra quyết định : Một cơ chế để quyết định kết quả cuối cùng của các phép tính toán, kết quả của N phiên bản sẽ được sử dụng như là kết quả đầu vào.
* Một chương trình giám sát : Một cấu trúc phần mềm được dùng để điều khiển N phiên bản và cơ chế ra quyết định.

Phần quan trọng nhất của phương pháp tiếp cận NVP là các đặc điểm kỹ thuật ban đầu. Các đặc điểm kỹ thuật này được coi là "cốt lõi" của phương pháp tiếp cận và được yêu cầu là rõ ràng nhưng cố gắng để áp đặt càng ít càng tốt các phương pháp thiết kế hoặc các thuật toàn phức tạp.

N phiên bản phần mềm được tạo ra độc lập từ các đặc điểm kỹ thuật ban đầu. Độc lập ở đây có nghĩa là chúng được phát triển bởi các đội kỹ sư khác nhau, bằng cách sử dụng các thuật toán và ngôn ngữ khác nhau thậm chí có thể khác nhau cả về trình biên dịch và hệ điều hành. Các đội lập trình cũng nên đảm bảo yếu tố đa dạng, tức là họ cần phải có nguồn gốc khác nhau, trình độ khác nhau, … .Các phiên bản N sẽ có chức năng tương đương và có giao diện giống với các phiên bản khác.

Cơ chế quyết định sử dụng kết quả của tất cả N phiên bản như kết quả đầu vào để ra quyết định xem kết quả nào sẽ là kết quả được trả ra cuối cùng. Cơ chế quyết định này thường là “một cử tri” đạt được sự đồng thuận từ đa số các phiên bản phần mềm. Việc so sánh có thể là giống nhau hoàn toàn (bỏ phiếu chính xác). Tuy nhiên có thể nhiều phần mềm không thể áp dụng được, khi giá trị đầu ra của N phiên bản có thể là dạng số và do đó có tính liên tục trong tự nhiên. Điều này đặt ra nhu cầu phải có một phạm vi cho phép sự khác biệt, tức là các phép so sánh có thể không hoàn toàn giống nhau (bỏ phiếu không chính xác).

Để cơ chế ra quyết định có thể hoạt động chính xác các kết quả đầu ra của N phiên bản phải được đồng bộ. Chương trình giám sát thực hiện việc giám sát tất cả các tương tác giữa các phiên bản phần mềm và cơ chế ra quyết định. Nó cũng xử lý một phần của cơ chế đồng bộ hóa trong đó đặt các phiên bản trong các trạng thái khác nhau của sự hoạt động. Trạng thái gốc của phiên bản là không hoạt động. Khi nó được gọi bởi chương trình giám sát nó được đưa vào trạng thái chời đợi. Phiên bản này ở trạng thái chờ đợi cho đến khi nó nhận được một yêu cầu dịch vụ, yêu cầu này chuyển phiên bản vào trạng thái hoạt động. Nếu bất kỳ điều kiện chấm dứt nào xảy ra, thực hiện sẽ được hủy bỏ và các phiên bản sẽ trở lại trạng thái không hoạt động. Nếu không, nó tạo ra một kết quả khi đạt được một điểm đồng bộ, thông báo cho chương trình giám sát kết quả là đã sẵn sàng, và trở lại trạng thái chờ đợi.

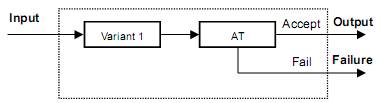


Hình 10. Các trạng thái của phiên bản.

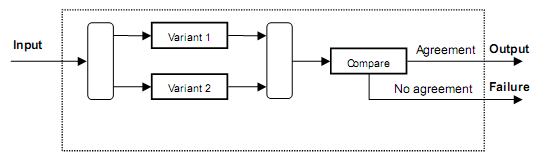
**2.2.2.3 N Self-Checking Programming (NSCP)**

Trong kĩ thuật N Selft-Checking Programming (NSCP) hệ thống được chia thành nhiều thành phần tự kiểm tra như là các biến thể khác nhau của phần mềm (Tương đương với các thành phần thay thế trong RB và các phiên bản trong NVP). Các thành phần này thực hiện song song. Một thành phần tự kiểm tra được thực hiện theo một trong hai cách:

* Một biến thể được kết hợp với một kiểm tra chấp nhận để kiểm tra kết quả của biến thể.
* Một cặp biến thể được kết hợp với một thuật toán so sánh để so sánh kết quả của hai biến thể.



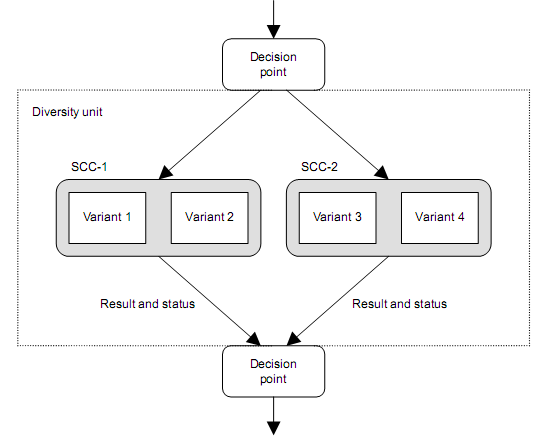
Hình 11.a : Thành phần tự kiểm tra với bộ kiểm thử chấp nhận

**

Hình 11.b : Thành phần tự kiểm tra gồm một cặp biến thể và bộ so sánh

Khả năng chịu lỗi được thực hiện bằng việc thực hiện song song N ≥ 2 các thành phần tự kiểm tra. Mỗi thành phần có trách nhiệm xác định liệu một kết quả trả ra có thể chấp nhận được. Thực tế là các thành phần tự kiểm tra được thực hiện song song làm tăng các yêu cầu của một cơ chế thống nhất đầu vào, và cũng cần thiết có một cơ chế dự phòng phần cứng. Một hệ thống thực hiện bằng cách sử dụng phương pháp tiếp cận NSCP thường có một cấu trúc nhiều lớp bao gồm các đơn vị đa dạng. Một đơn vị đa dạng là một mô tả chung của chương trình giữa hai điểm quyết định. Một điểm quyết định là một điểm trong chương trình quyết định một số kết quả sẽ là một kết quả được sử dụng bởi các tính toán sau đây.

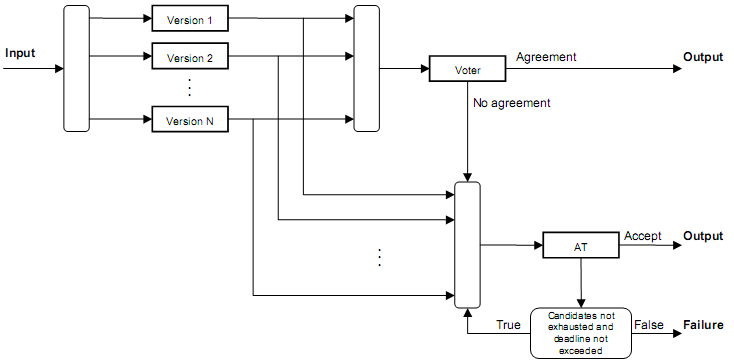
Trong một đơn vị đa dạng có một thành phần tự kiểm tra hoạt động. Các thành phần khác được gọi là “phần phụ”. Xử lý lỗi được thực hiện thông qua phát hiện lỗi và chuyển đổi kết quả. Nếu thành phần hoạt động không thể cung cấp một dịch vụ chính xác, các dịch vụ cung cấp sẽ ngay lập tức được chuyển sang các phần phụ.



Hình 12. Cấu trúc một đơn vị đa dạng.

**2.2.2.4 Consensus Recovery Block (CRB)**

Consensus Recovery Block – Khối phục hồi có sự đồng thuận (CRB) là sự kết hợp giữa hai phương pháp : RB và NVP. Khi bắt đầu hoạt động CRB giống như một hệ thống áp dụng phương pháp NVP, tức là N phiên bản sẽ được chạy song song và có sự biểu quyết. Một giả định cơ bản là trường hợp không có lỗi tương tự sẽ xảy ra. Nếu quá trình ra quyết định của NVP không thể thực hiện thì kết quả được trả ra từ N phiên bản sẽ được đưa vào bộ kiểm tra chấp nhận, bắt đầu với kết quả của phiên bản tốt nhất, phiên bản tốt thứ hai, … Kết quả đầu tiên vượt qua kiểm tra chấp nhật sẽ được coi là kết quả cuối cùng.



Hình 13. Kiến trúc cơ bản của CRB.

**2.2.2.5 So sánh RB và NVP :**

RB và NVP là hai kĩ thuật xây dựng phần mềm được sử dụng nhiều nhất. Mỗi kĩ thuật đều có những điểm mạnh yếu riêng của mình.

* Tính hiệu quả : Khi xây dựng 2 kĩ thuật trên cho một nhiệm vụ thực hiện trong 10 tiếng các kĩ sư lập trình thấy rằng RB có hiệu quả rõ ràng hơn nhiều so với NVP.
* Độ tin cậy : Với cả RB và NVP xác suất của một sự thất bại chủ yếu được quy định bởi các lỗi liên quan giữa các thành phần. Trong RB một lỗi liên quan giữa các thành phần chính và phụ khó có thể gây ra một thất bại. Lỗi liên quan giữa các thành phần phụ và bộ kiểm thử chấp nhận có thể gây ra thất bại nếu bộ kiểm thử từ chối kết quả từ thành phần chính. Với NVP xác suất của sự thất bại có thể cao gấp ba lần so với RB.
* Hiệu năng : RB có hiệu năng cao hơn so với NVP. Thời gian của RB bị chi phối bởi sự kết hợp của các thành phần chính và phụ với bộ kiểm thử chấp nhận là chủ yếu. Thời gian trung bình của NVP lại bị kéo dài vì các biến thể khi đồng bộ hóa yêu cầu hệ thống phải đợi các biến thể chậm nhất.

# CHƯƠNG 3. Xây dựng mô hình Master-Slave-Client

Như đã đề cập trong Chương 1, kiến trúc hệ thống sẽ được xây dựng theo mô hình Master – Slave – Client gồm ba thành phần chính như sau :

* Master Server : Một server thực hiện nhiệm vụ điều phối và quản lý các Slave Server. Master thực hiện việc kiểm tra thông số và tình trạng hiện thời của các Slave Server và gửi thông báo cho người sử dụng.
* Slave Server : Các Slave server kết nối và trao đổi thông tin trực tiếp với Client, đóng vai trò chính trong việc xử lý các thông điệp từ Client và xử lý cơ sở dữ liệu. Các yêu cầu trong game khi được client yêu cầu, slave server sẽ đóng vai trò xử lý chính.
* Client : Là các thiết bị di động của người sử dụng. Thực hiện việc truyền dữ liệu và xử lý thông tin do server gửi về, các hình ành và giao diện cũng sẽ được client hiển thị tới người sử dụng.

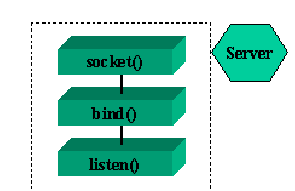
## 3.1 Mô hình kết nối client – server sử dụng socket

Socket là phương tiện hiệu quả để xây dựng các ứng dụng theo kiến trúc Client-Server. Các ứng dụng trên mạng Internet như Web, mail, FTP là các ví dụ điển hình. Phần này trình bày các bước cơ bản trong việc xây dựng mô đun kết nối cơ bản Client-Server sử dụng Socket làm phương tiện giao tiếp theo TCP và UDP. Dựa vào việc phân tích ưu nhược điểm của các loại giao tiếp để quyết định áp dụng một loại giao tiếp phù hợp.

### 3.1.1 Kết nối theo TCP

**3.1.1.1 Giai đoạn 1**

Server tạo Socket, gán số hiệu cổng và lắng nghe yêu cầu nối kết



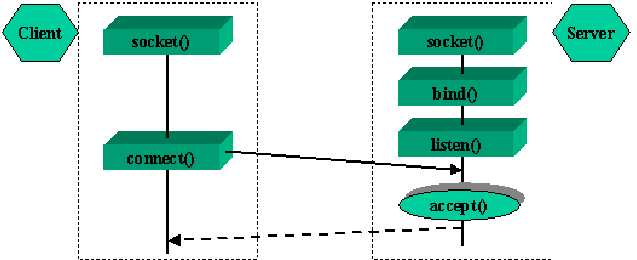
Hình 14: Server tạo Socket theo TCP

* *socket()*: Server yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển.
* *bind()*: Server yêu cầu gán số hiệu port cho socket.
* *listen()*: Server lắng nghe các yêu cầu nối kết từ các client trên cổng đã được gán.

Server sẳn sàng phục vụ Client.

**3.1.1.2 Giai đoạn 2**

Client tạo Socket, yêu cầu thiết lập một nối kết với Server



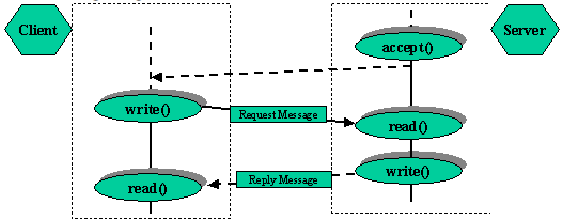
Hình 15: Client tạo socket và thiết lập kết nối theo TCP

Việc thiết lập kết nối được tiến hành theo các bước như sau:

* *socket()*: Client yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển, thông thường hệ thống tự động gán một số hiệu cổng còn rãnh cho socket của Client. Client cần được biết địa chỉ của socket và số hiệu cổng mà socket sẵn sàng chấp nhận kết nối.
* *connect()*: Client gửi yêu cầu nối kết đến server có địa chỉ IP và Port xác định.
* *accept()*: Server chấp nhận nối kết của client, khi đó một kênh giao tiếp ảo được hình thành,Client và server có thể trao đổi thông tin với nhau. Kênh giao tiếp bao gồm luồng đọc DataInputStream và luồng ghi DataOutputStream.

**3.1.1.3 Giai đoạn 3**

Trao đổi thông tin giữa Client và Server



Hình 16: Trao đổi thông tin giữa client và server qua socket theo TCP

* Sau khi chấp nhận yêu cầu nối kết, thông thường server thực hiện lệnh *read()* để nghẽn cho đến khi có thông điệp yêu cầu (Request Message) từ client gởi đến.
* Server phân tích và thực thi yêu cầu. Kết quả sẽ được gởi về client bằng lệnh *write()*.
* Sau khi gởi yêu cầu bằng lệnh *write()*, client chờ nhận thông điệp kết quả (ReplyMessage) từ server bằng lệnh *read()*.

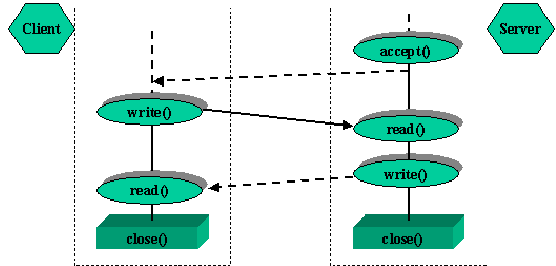
Trong giai đoạn này, việc trao đổi thông tin giữa Client và Server phải tuân thủ giao thức của ứng dụng (Dạng thức và ý nghĩa của các thông điệp, qui tắc bắt tay, đồng bộ hóa, ... ). Thông thường Client sẽ là người gởi yêu cầu đến Server trước.

Nếu chúng ta phát triển ứng dụng theo các Protocol đã định nghĩa sẳn, chúng ta phải tham khảo và tuân thủ đúng những qui định của giao thức. Ngược lại, nếu chúng ta phát triển một ứng dụng Client-Server riêng của mình, thì công việc đầu tiên chúng ta phải thực hiện là đi xây dựng Protocol cho ứng dụng.

**3.1.1.4 Giai đoạn 4**

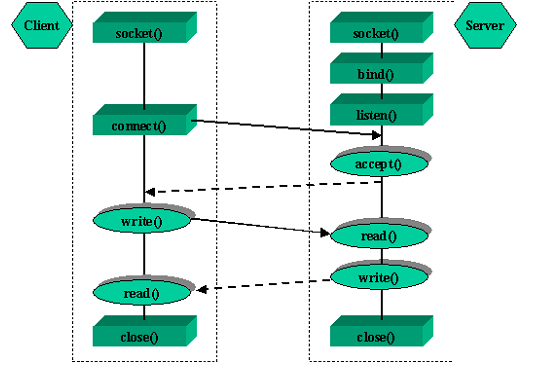
Kết thúc phiên làm việc

* Các câu lệnh *read()*, *write()* có thể được thưc hiện nhiều lần (ký hiệu bằng hình elipse trong hình 17).
* Kênh kết nối ảo sẽ bị xóa khi Server hoặc Client đóng kết nối socket bằng lệnh *close()*.



Hình 17: Kết thúc phiên kết nối socket dùng TCP.

Như vậy toàn bộ quá trình kết nối giữa client và server theo TCP diễn ra như sau :



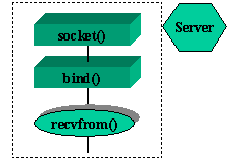
Hình 18: Toàn bộ quá trình kết nối client – server dùng socket theo TCP.

### 3.1.2 Kết nối theo UDP

**3.1.2.1 Giai đoạn 1**

Server tạo Socket - gán số hiệu cổng

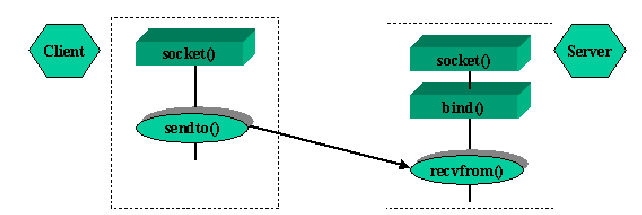
* socket(): Server yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển.
* bind(): Server yêu cầu gán số hiệu cổng cho socket.



Hình 19: Server tạo socket theo UDP.

**3.1.2.2 Giai đoạn 2**

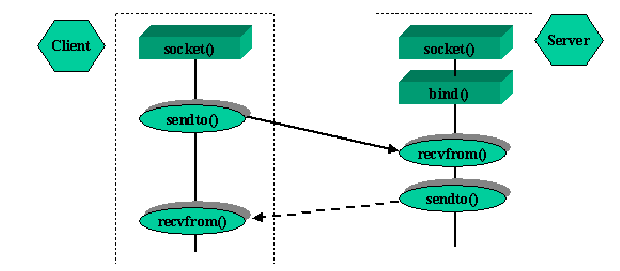
Client tạo socket



Hình 20: Client tạo socket theo UDP.

**3.1.2.3 Giai đoạn 3**

Trao đổi thông tin giữa client và server



Hình 21: Trao đổi thông tin giữa client và server theo UDP.

Sau khi tạo Socket xong, Client và Server có thể trao đổi thông tin qua lại với nhau thông qua hai hàm sendto() và recvfrom(). Đơn vị dữ liệu trao đổi giữa Client và Server là các Datagram Package (Gói tin thư tín).

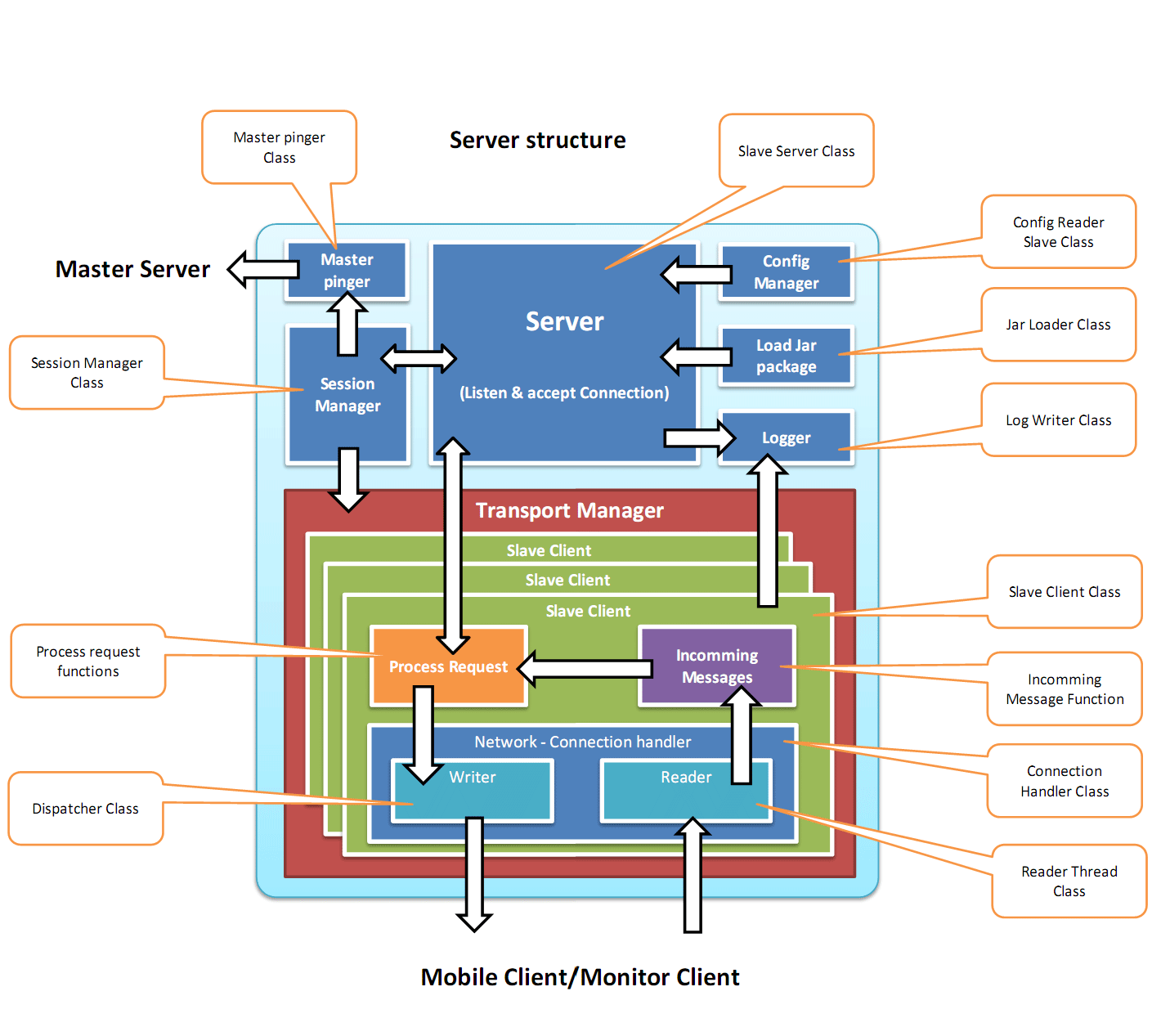
## 3.2 Xây dựng server sử dụng socket trong Java

Hệ thống em xây dựng dựa trên nền tảng ngôn ngữ Java. Trong Java có hỗ trợ đầy đủ các phương thức kết nối sử dụng socket, trong đó có một thành phần rất quan trọng để xây dựng server sử dụng kết nối socket là SocketServer. Lớp ServerSocket có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng ServerSocket mới, các phương thức để lắng nghe các liên kết trên một cổng xác định, và các phương thức trả về một Socket khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu. Các server sử dụng socket thường được xây dựng theo cấu trúc :

1. Một ServerSocket mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor ServerSocket. Cổng được quy định trên mỗi server, việc khởi tạo ServerSocket cũng giống như việc server mở cổng để sẵn sàng nhận các kết nối.
2. ServerSocket lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức *accept().* Phương thức *accept()* phong tỏa cho tới khi một client thực hiện một liên kết tới server, phương thức *accept()* trả về một đối tượng Socket được dùng để thiết lập liên kết giữa client và server.
3. Tùy thuộc vào kiểu server muốn xây dựng chúng ta gọi các luồng đọc ghi dữ liệu thích hợp. Nếu server chỉ có nhiệm vụ là thu nhận thống tin phương thức *getInputStream()* sẽ được gọi, nếu server chỉ có nhiệm vụ gửi thông tin về client thì phương thức *getOutputStream()* được sử dụng, trong trường hợp server sẽ vừa nhận yêu cầu từ client vừa gửi thông tin thì cả hai phương thức sẽ được gọi để nhận các luồng vào ra để truyền tin với client.
4. Server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết. Việc xây dựng giao thức truyền tin giữa server và client là rất quan trọng. Giao thức truyền tin cần phải định nghĩa định dạng các gói tin, phân loại các yêu cầu, phân loại dữ liệu gửi lên và dữ liệu trả về để tránh mất mát gói tin và xử lý sai yêu cầu.
5. Server, client hoặc cả hai ngắt liên kết khi phiên làm việc kết thúc.
6. Server trở về bước hai và đợi liên kết tiếp theo.

Dựa vào kiến trúc hệ thống đã phân tích ở chương 3 và cấu trúc server sử dụng socket như đã trình bày ở trên ở phần sau em sẽ trình bày phần thiết kế hệ thống chi tiết. Các thành phần sẽ được thiết kế bao gồm : Master Server, Slave Server, Client.

## 3.3 Kiến trúc server



Hình 22. Kiến trúc server

### 3.3.1 Mô đun quản lý server

Mô đun gồm các thành phần phụ trách việc quản lý hoạt động, thiết lập cấu hình và các trạng thái của server. Mô đun này cũng phụ trách việc thiết lập kết nối với client khi có yêu cầu kết nối.

**3.3.1.1 Slave Server**

Là thành phần quan trọng nhất của mô đun, thực hiện việc điều khiển hoạt động của tất cả các thành phần trong mô đun và cả các thành phần khác trong Server. Các nhiệm vụ chính của thành phần này :

* Khởi tạo các thông số cấu hình cho server (Địa chỉ IP, số hiệu cổng kết nối, địa chỉ, tài khoản và mật khẩu của cơ sở dữ liệu, …), thực hiện kết nối với cơ sở dữ liệu, kết nối với master server.
* Lắng nghe và thực hiện kết nối với client. Khi slave server chạy sẽ mở một cổng để các client có thể thực hiện kết nối. Khi có một yêu cầu kết nối được gửi đến, server sẽ tạo ra một đối tượng “slave client” phụ trách việc trao đổi thông tin với client, đồng thời thiết lập các thông số cần thiết để quản lý phiên làm việc của client.
* Thao tác với cơ sở dữ liệu.

**3.3.1.2 Config Manager**

Mỗi server đều có một loạt các thông số cấu hình cho các hoạt động của mình. Thành phần Config Manager phụ trách việc quản lý các thông số cấu hình của server. Các thông số cấu hình của server có thể bao gồm: Địa chỉ các thư mục cần thiết; tên server; địa chỉ IP; số hiệu cổng kết nối; địa chỉ master; địa chỉ, tài khoản và mật khẩu kết nối với cơ sở dữ liệu; … Để đảm bảo tính linh động và dễ dàng trong việc triển khai hệ thống các thông số cấu hình trên sẽ được lưu trong một file cấu hình. Mỗi khi khởi chạy, server sẽ yêu cầu thành phần config manager đọc file cấu hình và trả ra các thông số cần thiết. Khi có thông số nào đó cần thay đổi chúng ta chỉ cần chỉnh sửa lại file cấu hình.

**3.3.1.3 Logger**

Thành phần Logger phụ trách việc ghi lại “log” của server. Khi server hoạt động, luôn có rất nhiều các giao dịch được thực hiện, việc ghi lại trạng thái của server là cần thiết để dự phòng trường hợp giao dịch xảy ra sai sót. Khi một giao dịch nào đó giữa client và server có xảy ra sai sót dựa vào các thông tin dự phòng được ghi lại chúng ta có thể tiến hành xử lý và phục hồi thông tin. Việc ghi “log” cũng giúp chúng ta có thể theo dõi được trạng thái của hệ thống từ đó đưa ra những biện pháp xử lý thích hợp khi gặp lỗi.

**3.3.1.4 Master Pinger**

Đây là thành phần phụ trách việc kết nối với master slave của slave server. Sau một thời gian hoạt động nhất định (đã được quy định trong các thông số cấu hình) slave server sẽ thực hiện việc “báo cáo” tình trạng hiện tại của mình với master server.

**3.3.1.5 Session Manager**

Khi có một kết nối từ client được thiết lập, server sẽ khởi tạo một phiên làm việc để quản lý việc kết nối giữa client và server. Khi kết nối kết thúc thành phần session manager cũng sẽ phụ trách việc đóng kết nối và dọn dẹp tài nguyên.

### 3.3.2 Mô đun quản lý giao vận

Mô đun này thực hiện các chức năng trao đổi và xử lý thông tin trực tiếp với client. Mô đun bao gồm các thành phần chính :

**3.3.2.1 Slave Client**

Khi một kết nối với client được thiết lập, một đối tượng slave client được tạo ra để phụ trách việc trao đổi thông tin với client. Các thông tin được gửi từ client sẽ được phân tích thành các gói tin sau đó dựa vào từng loại gói tin sẽ được ra các phương thức xử lý thích hợp và kết quả trả ra sẽ được gửi cho client.

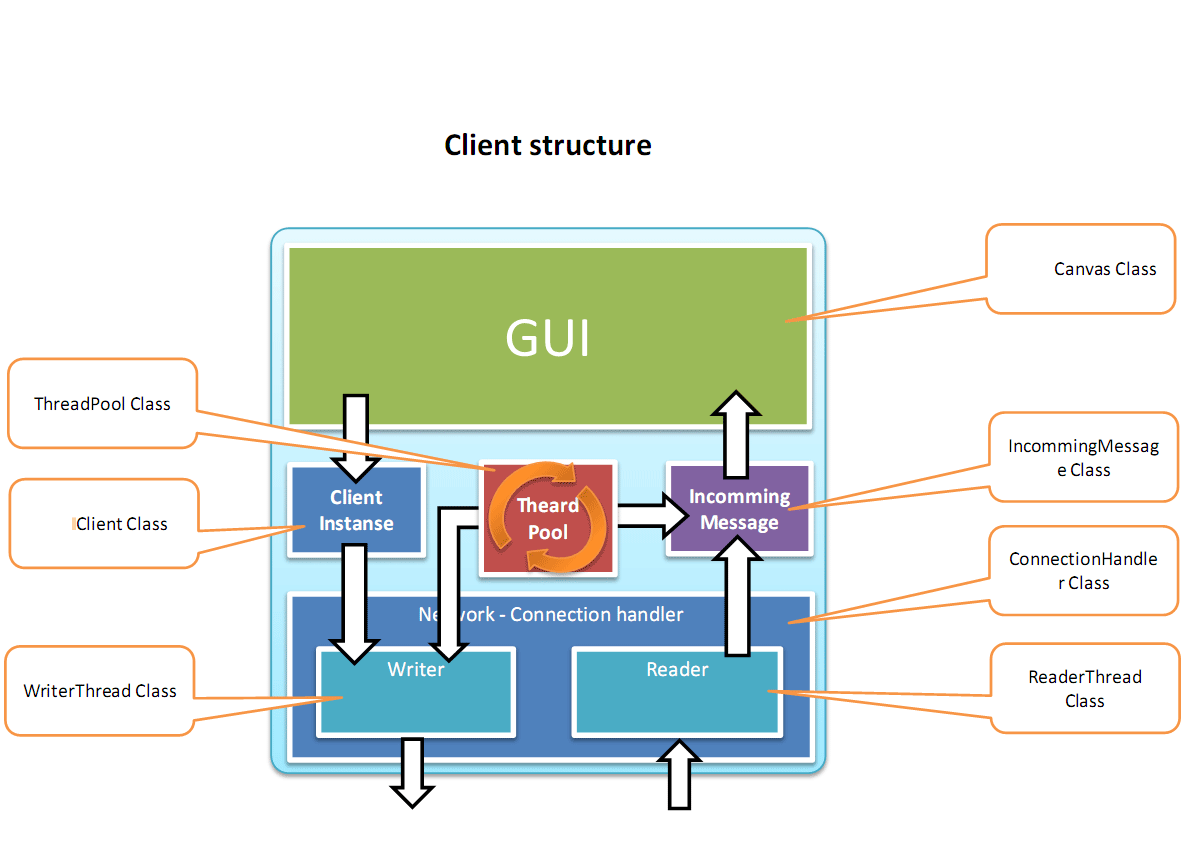
**3.3.2.2 Connection Handler**

Connection Handler phụ trách việc duy trì kết nối của client và server. Nó gồm 2 thành phần quan trọng

* Reader: Phụ trách đọc các thông tin được truyền lên từ client.
* Writer: Phụ trách gửi các thông tin đã được server xử lý về cho client.

Cơ chế đọc ghi dữ liệu có sử dụng đến “queue” để lưu trữ dữ liệu. Các gói tin khi được gửi lên sẽ được đẩy vào queue, gói tin nào đến trước sẽ được xử lý trước. Tương tự các gói tin khi được gửi tới client cũng sẽ được đẩy vào queue, gói tin nào được xử lý trước sẽ được gửi tới client trước. Connection Handler có nhiệm vụ đảm bảo việc nhận và gửi thông tin với client được thực hiện thông suốt và tránh tình trạng mất mát thông tin. Khi có sự cố xảy ra với kết nối Connection Handler cũng sẽ thông báo với server để đưa ra các phương thức giải quyết sự cố.

## 3.4 Kiến trúc client



Hình 23. Kiến trúc client

Kiến trúc Client bao gồm các thành phần chính như sau :

### 3.4.1 GUI – Giao diện người dùng

Đây là thành phần chính của phía Client. Với các ứng dụng game trên di động thành phần giao diện đồ họa sẽ hiển thị các hình ảnh trong game, các menu tương tác với người sử dụng.

### 3.4.2 Connection Handler

Giống như ở trên server, phía client cũng có các phương thức phụ trách việc đọc, ghi và xử lý dữ liệu nhận được từ server.

**3.4.2.1 Writer :**

Writer sẽ thực hiện việc gửi các thông điệp lên server. Các thông điệp này bao gồm mã yêu cầu và các thông tin cần thiết từ phía Client. Chú ý rằng vì đường truyền của các thiết bị di động là chậm nên các dữ liệu trước khi gửi cần được tối ưu hóa.

**3.4.2.2 Reader :**

Nhiệm vụ của Reader là đọc dữ liệu tương ứng với các yêu cầu đã được server xử lý. Reader sẽ bóc tách các gói tin, phân loại chúng để Client xử lý. Điểm mấy chốt là Reader phải đảm bảo không để mất gói tin khi nhận từ server về.

**3.4.2.3 Incoming Messages**

Các thông tin từ server được Reader gửi về cần phải qua quá trình xử lý và bóc tách trước khi hiển thị ra GUI. Thành phần Incoming Messages sẽ thực hiện nhiệm vụ này.

## 3.5 Kĩ thuật chịu lỗi áp dụng cho mô đun kết nối

### 3.5.1 Mô đun kết nối

Như đã trình bày trong hai phần 3.1 và 3.2 các thành phần của Connection Handler (ở cả server và client) sẽ làm nhiệm vụ khởi tạo và duy trì kết nối. Việc kết nối giữa client và server sẽ được thực hiện theo trình tự như sau:

Client ban đầu sẽ kết nối tới Master để khởi tạo kết nối. Từ master, client sẽ lấy được thông tin về người chơi về thông tin về các slave server hiện đang chạy. Như đã nói master và slave luôn giữ một sự kết nối, các thông tin về trạng thái của slave luôn được master cập nhật và gửi thông báo cho client khi có yêu cầu kết nối. Thông tin về slave bao gồm: Tên slave server, địa chỉ, số hiệu cổng, số user đang đăng nhập, tình trạng (còn hoạt động bình thường, đã “chết” hoặc mất kết nối, …).

Vấn đề đặt ra ở đây là khi master xảy ra lỗi phải ngừng hoạt động, hoặc kết nối của master bị ngắt (do mất mạng, mất điện, …) thì toàn bộ các kết nối tới hệ thống sẽ không thể thực hiện được, trong khi các slave server vẫn hoạt động bình thường và vẫn có thể đáp ứng được các nhu cầu chơi game của người chơi. Để khắc phục tình trạng này chúng ta cần phải áp dụng các kĩ thuật chịu lỗi đã được trình bày ở trên và kĩ thuật được lựa chọn trong trường hợp này là kĩ thuật Recovery Block.

### 3.5.2 Áp dụng kĩ thuật chịu lỗi Recovery Block

Với kĩ thuật Recovery Block chúng ta sẽ xây dựng những cách thức kết nối hoàn toàn khác nhau từ client lên hệ thống. Hai cách thức đó là : Kết nối qua socket tới master và kết nối qua HTTP để lấy thông tin hệ thống.

* Kết nối qua socket tới master : Đây sẽ là phương thức kết nối chính được sử dụng như đã trình bày ở các phần trước.
* Kết nối qua HTTP : Khi việc kết nối tới master không thể thực hiện, để client vẫn có thể lấy được thông tin về các slave để thực hiện kết nối chúng ta có thể sử dụng một giao thức kết nối khác là HTTP. Khi đó sẽ có một địa chỉ http tại đó lưu trữ các thông tin của các slave đang hoạt động trong hệ thống, client thông qua kết nối HTTP sẽ kết nối và lấy được các thông tin của hệ thống từ đó có thể kết nối trực tiếp tới các slave để tham gia trò chơi.

Bộ kiểm thử chấp nhận được thiết kế để kiểm tra các thông tin của hệ thống mà client nhận được liệu có hợp lệ (Danh sách các slave server, địa chỉ của mỗi slave, số hiệu cổng của mỗi slave).

Kết nối thành công

true

false

Khôi phục trạng thái hệ thống

Kết nối thất bại

Vẫn còn phương thức kết nối

Kiểm thử chấp nhận

HTTP

Socket

SWITCH

Lưu trữ trạng thái hệ thống

Yêu cầu kết nối

Hình 24. Mô đun kết nối ứng dụng kĩ thuật Recovery Block.

# CHƯƠNG 4. THIẾT KẾ HỆ THỐNG

Trong chương 3 em đã trình bày về mô hình master-slave-client với đầy đủ các thành phần cần thiết để xây dựng một hệ thống game online trên di động. Trong chương này dựa vào mô hình đã được xây dựng em sẽ thiết kế một hệ thống game online, các thiết kế sẽ được tập trung vào hai thành phần chính là server (bao gồm cả master và slave) và client.

## 4.1 Giới thiệu về sản phẩm

Trong quá trình nghiên cứu thị trường game online trên các thiết bị di động em nhận thấy rằng đa số các phần mềm hiện tại đều tập trung vào 2 mảng là : Game kiếm hiệp và game casino. Các dòng game này hiện tại đã dần dần bão hòa do có rất nhiều nhà sản xuất game phát triển hoặc mua lại các game nước ngoài. Người sử dụng cần có một game online mới, đảm bảo nhu cầu thư giãn và không gây nhàm chán. Sau một thời gian tìm hiểu em nhận thấy rằng các game show trí tuệ trên truyền hình mặc dù đã lên sóng từ rất lâu nhưng vẫn rất thu hút khán giả vì vậy em đã quyết định phát triển một game online theo cách thức thi đấu trí tuệ.

Sản phẩm em lựa chọn để xây dựng trong đợt thực tập này là một game online trên di động với tên “Đấu trường trí tuệ”. Trong game người chơi sẽ tham gia thì đấu với các người chơi khác bằng các câu hỏi thuộc nhiều lĩnh vực khác nhau. Luật chơi game cũng khá đơn giản: Mỗi lượt chơi người chơi sẽ tham gia vào một bàn có tối đa là 8 người chơi. Người trả lời đúng sẽ được điểm tùy theo thời gian trả lời nhanh hay chậm, người trả lời sai sẽ không được điểm. Mỗi lượt chơi có 10 câu hỏi, sau 10 câu hỏi người chơi nào có tổng số điểm cao nhất sẽ là người chiến thắng. Mỗi lượt chơi đều có một lượng tiền cược nhất định do người chủ phòng quy định. Những người thua sẽ bị trừ tiền bằng số tiền cược, người thắng sẽ nhận được tổng số tiền cược trong bàn. Em sẽ xây dựng game trên hai nền tảng di động phổ biến hiện nay là J2ME và Android. Việc xây dựng hệ thống hoàn toàn tuân theo kiến trúc đã phân tích ở trên.

## 4.2 Các chức năng của game

### 4.2.1 Các chức năng cơ bản

Game được xây dựng là hệ thống game online vì vậy mỗi người sử dụng cần có một tài khoản trong hệ thống. Chức năng đăng ký sẽ giúp người dùng có thể đăng ký một tài khoản để đăng nhập vào hệ thống. Tài khoản gồm các thông tin rất đơn giản như : Tên tài khoản, mật khẩu, giới tính. Tên đăng nhập phải là duy nhất vì nó sẽ được sử dụng để phân biệt các người sử dụng. Trước khi đăng kí người sử dụng sẽ cần được biết các điều khoản sử dụng trong game. Khi người dùng chấp nhận điều khoản sử dụng và điền đầy đủ thông tin đăng ký họ sẽ gửi đăng ký. Khi server nhận được thông tin đăng ký, server sẽ kiểm tra các thông tin này liệu có hợp lệ hay không, nếu hợp lệ hệ thống sẽ tạo 1 tài khoản cho người sử dụng, cấp cho họ 1 “ID” và lưu trữ các thông tin vào cơ sở dữ liệu, nếu không hợp lệ gửi lại thông báo lỗi tới client để người sử dụng đăng ký lại. Khi người sử dụng đăng ký thành công, họ có thể dùng tài khoản và mật khẩu đã đăng ký để đăng nhập vào hệ thống và sẵn sàng tham gia và những trò chơi trong hệ thống.



Hình 25: Biểu đồ use case game “Đấu trường trí tuệ”.

Sau khi đã đăng nhập vào hệ thống người chơi có thể chọn lựa các phòng chơi, mỗi phòng chơi có quy định số lượng người chơi nhất định. Các thông tin về số lượng người tối đa trong một phòng, số người chơi hiện tại trong phòng sẽ được server cập nhật cho client để hiển thị tới người sử dụng. Khi phòng chơi đã đầy người chơi phải lựa chọn phòng chơi khác để vào chơi. Số lượng phòng chơi và số người tối đa trong phòng chơi đều được server gửi thông tin cho client, điều này giúp các phiên bản client có sự thống nhất.

Khi vào được phòng, người chơi sẽ lựa chọn bàn chơi để vào chơi. Mỗi bàn chơi có 8 vị trí, người vào đầu tiên của bản sẽ là chủ phòng, khi bàn đã đầy người chơi không thể vào bàn được nữa. Bàn chơi có thể đang ở trạng thái đang chờ hoặc đang chơi, bàn có đặt mật khẩu hay không, số tiền cược của bàn. Những thông tin này cũng được server gửi tới các client khi bất cứ bàn nào có sự thay đổi trạng thái.

Sau khi vào bàn chơi người chơi có thể chơi luyện tập hoặc là thi đấu với những người chơi khác. Ở chế độ luyện tập người chơi sẽ trả lời 15 câu hỏi theo độ khó tăng dần. Khi vượt qua các mốc câu hỏi là 5, 10, 15 người chơi sẽ nhận được các phần thưởng tương ứng với các mốc câu hỏi. Câu hỏi và câu trả lời sẽ được server lấy trong cơ sở dữ liệu câu hỏi và gửi về cho client. Ở chế độ thi đấu người chơi sẽ tham gia trả lời các câu hỏi với những người chơi khác. Mỗi lượt chơi có 10 câu hỏi, sau 10 câu hỏi hệ thống sẽ thống kế kết quả và ai có tổng điểm cao nhất sẽ là người chiến thắng. Câu hỏi và đáp án hệ thống sẽ gửi về cho tất cả các người chơi cùng một lúc. Kết quả sau mỗi câu hỏi cũng sẽ được hệ thống tính toán, sắp xếp và gửi về cho người chơi.

Ngoài các chức năng chính phục vụ cho việc chơi game như đã kể trên hệ thống còn bao gồm các chức năng hỗ trợ khác như sau:

* Nạp tiền: Tiền trong game là một loại tiền ảo được tính bằng một đơn vị gọi là “xu”. Tiền ảo sẽ được dùng để trao đổi, làm phần thưởng và làm tiền cược trong game. Để có thể nạp tiền ảo cho tài khoản trong hệ thống, người sử dụng có thể sử dụng các hình thức thanh toán như là qua tin nhắn SMS hoặc thẻ cào (Đây đều là các hình thức thanh toán khá phổ biến trên di động).
* Mua item: Người chơi có thể sử dụng tiền ảo của mình không chỉ trong việc thi đấu mà còn có thể mua sắm các “item” phụ trợ trong game.
* Tin nhắn: Trong game mỗi người chơi đều có một hộp thư, chứa các tin nhắn từ bạn bè, tin thông báo từ hệ thống hoặc lời mời kết bạn.
* Bạn bè: Trong game người chơi có thể gửi lời mời kết bạn tới người chơi khác. Bạn bè có thể gửi tin nhắn, gửi quà và xem thông tin của nhau.

### 4.2.2 Các chức năng trong bàn chơi

Bàn chơi là nơi mà người chơi thực hiện các chức năng chính của trò chơi như thi đấu với những người chơi khác hoặc luyện tập với các câu hỏi có độ khó tăng dần vì vậy các chức năng của người chơi ở trong bàn chơi là khá phức tạp. Trong game một bàn chơi được thiết kế với 8 vị trí, tức là cùng một lúc có thể có 8 người chơi ở trong bàn chơi. Khi người chơi ở trong bàn chơi, hệ thống cần đáp ứng những chức năng sau:

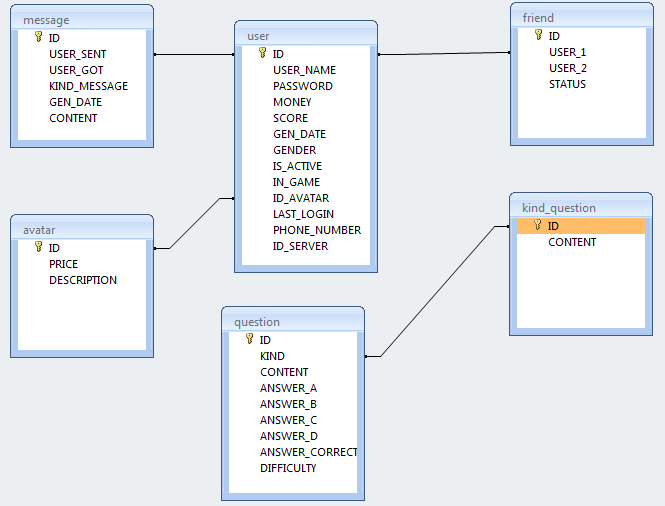
* Trong bàn chơi người chơi có thể là chủ bàn hoặc là người chơi.
* Chủ bàn là người có quyền quyết định trong bàn với các chức năng như : Đặt cược, đặt mật khẩu của bàn chơi, loại một người chơi không mong muốn ra khỏi bàn, mời người chơi khác vào bàn, có quyền bắt đầu ván chơi khi các người chơi khác đã sẵn sàng.
* Người chơi bình thường vào bàn và phải sẵn sàng để bắt đầu ván chơi, chỉ khi tất cả người chơi trong bàn đã sẵn sàng thì ván chơi mới có thể bắt đầu.
* Ngoài ra khi bàn đang chơi mà người chơi tham gia vào phòng thì sẽ ở chế độ xem, khi đó người chơi không thể thực hiện các thao tác khi trò chơi diễn ra, người chơi chỉ có thể xem và đợi khi ván chơi hiện tại kết thúc để tham gia chơi.

Các chức năng trong bàn chơi được thể hiện rõ trong biểu đồ phía dưới.



Hình 26: Các chức năng trong bàn chơi.

## 4.3 Cơ sở dữ liệu



Hình 27: Cơ sở dữ liệu của game.

Sau khi phân tích chức năng và yêu cầu của game, phần này em xin trình bày về cơ sở dữ liệu của hệ thống, cơ sở dữ liệu của hệ thống bao gồm 6 bảng như sau:

* Bảng *USER*: Chứa tất cả các thông tin của người chơi trong game như: Tên tài khoản, mật khẩu, tiền, số điện thoại, …
* Bảng *AVATAR*: Chứa các thông tin của các item mà người chơi sử dụng.
* Bảng *QUESTION*: Chính là ngân hàng câu hỏi được sử dụng trong game.
* Bảng *KIND\_QUESTION*: Chứa các lĩnh vực của ngân hàng câu hỏi.
* Bảng *MESSAGE*: Chứa các tin nhắn của người chơi, các tin nhắn có thể là tin của hệ thống hoặc tin của bạn bè.
* Bảng *FRIEND*: Lưu trữ tình trạng quan hệ bạn bè của người chơi trong game.

Sau đây em xin mô tả chi tiết các bảng của cơ sở dữ liệu:

- Bảng USER :

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Mô tả** |
| ID | Lưu trữ id của người chơi |
| USER\_NAME | Tên tài khoản mà người chơi đăng ký |
| PASSWORD | Mật khẩu của tài khoản do người chơi đăng ký |
| MONEY | Tiền có trong tài khoản của người chơi |
| SCORE | Điểm tích lũy của người chơi |
| GEN\_DATE | Lưu trữ ngày người chơi tạo tài khoản trên hệ thống |
| GENDER | Giới tính của người chơi |
| IS\_ACTIVE | Tình trạng tài khoản đã kích hoạt hay chưa |
| IN\_GAME | Lưu trữ trạng thái của người chơi đã đăng nhập hay chưa |
| ID\_AVATAR | Hình ảnh đại diện mà người chơi hiện đang sử dụng |
| LAST\_LOGIN | Lần đăng nhập cuối cùng của người chơi |
| PHONE\_NUMBER | Số điện thoại của người chơi |
| ID\_SERVER | Lưu trữ số hiệu của server hiện tại người chơi đang đăng nhập. |

Bảng 1: Mô tả bảng USER

- Bảng *AVATAR*:

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Mô tả** |
| ID | Lưu trữ id của avatar |
| PRICE | Lưu giá tiền của avatar (vì avatar phải mua trong cửa hàng) |
| DESCRIPTION | Chứa mô tả về avatar |

Bảng 2: Mô tả bảng AVATAR.

- Bảng *QUESTION:*

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Mô tả** |
| ID | Lưu trữ id của câu hỏi |
| KIND | Lưu trữ lĩnh vực của câu hỏi |
| CONTENT | Lưu trữ nội dung của câu hỏi |
| ANSWER\_A | Đáp án A của câu hỏi |
| ANSWER\_B | Đáp án B của câu hỏi |
| ANSWER\_C | Đáp án C của câu hỏi |
| ANSWER\_D | Đáp án D của câu hỏi |
| ANSWER\_CORRECT | Phương án trả lời đúng |
| DIFFICULTY | Độ khó của câu hỏi |

Bảng 3: Mô tả bảng QUESTION.

- Bảng *MESSAGE*:

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Mô tả** |
| ID | Lưu trữ id của message |
| USER\_SENT | Chứa tên user gửi tin |
| USER\_GOT | Chứa tên user được nhận tin |
| KIND\_MESSAGE | Chứa loại tin nhắn, tin nhắn có thể là : Tin hệ thống thông báo tới người chơi, tin nhắn thông thường của bạn bè, tin kết bạn. |
| GEN\_DATE | Thời gian tin được gửi |
| CONTENT | Nội dung tin nhắn |

Bảng 4: Mô tả bảng MESSAGE.

- Bảng *FRIEND:*

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Mô tả** |
| ID | Lưu trữ id của tình trạng bạn bè |
| USER\_1 | Chứa tên của người chơi 1 |
| USER\_2 | Chứa tên của người chơi 2 |
| STATUS | Tình trạng quan hệ bạn bè của hai người chơi |

Bảng 5: Mô tả bảng FRIEND.

- Bảng *KIND\_QUESTION:*

|  |  |
| --- | --- |
| **Trường dữ liệu** | **Mô tả** |
| ID | Lưu trữ id của loại câu hỏi |
| CONTENT | Chứa tên của loại câu hỏi. Loại câu hỏi có thể là: Toán học, Văn học, Lịch sử, Địa lý,… |

Bảng 6: Mô tả bảng KIND\_QUESTION.

Với cơ sở dữ liệu của game hai bảng quan trọng nhất chính là USER và QUESTION. Bảng USER chứa toàn bộ các thông tin của người chơi đăng ký trong game, còn bảng QUESTION chứa toàn bộ các ngân hàng câu hỏi của hệ thống, việc tạo hứng thú chơi game và cảm giác mới lạ cho người chơi phụ thuộc rất nhiều vào ngân hàng câu hỏi phải thật chính xác và phong phú.

Trong phần tiếp theo em sẽ trình bày về thiết kết của server và client. Về phía server em sẽ thiết kế chi tiết cho Master Server và Slave Server. Theo mô hình server đã được trình bày ở chương 3 thì master và slave đều có những mô đun khá giống nhau, nhưng do sự khác biệt về chức năng nên các mô đun này được cài đặt một cách khác nhau. Ngoài ra mỗi server đều có những thành phần khác biệt phục vụ các chức năng của riêng mình. Chẳng hạn ở slave server có các thành phần xử lý chức năng của game phức tạp hơn so với master, ngoài ra ở slave server còn có mô đun chịu lỗi được cài đặt để đề phòng trường hợp master gặp sự cố vẫn có thể khởi tạo kết nối trực tiếp với client.

## 4.4 Master Server



Hình 28. Biểu đồ lớp của Master server.

Dựa vào kiến trúc của server đã được trình bày ở Chương 3, trong phần này em xin được trình bày chi tiết thiết kế của server Master, xây dựng biểu đồ lớp cho các thành phần chính của server Master. Master server bao gồm ba thành phần quan trọng nhất như sau : Mô đun quản lý server, mô đun kết nối, mô đun cơ sở dữ liệu.

### 4.4.1 Mô đun quản lý server

Mô đun gồm các thành phần phụ trách việc quản lý hoạt động, thiết lập cấu hình và các trạng thái của server. Mô đun này cũng phụ trách việc thiết lập kết nối với client khi có yêu cầu kết nối đồng thời quản lý trạng thái của các “user” đang hoạt động trên hệ thống.

**4.4.1.1 Master Server**

**a. Các thành phần chính của MasterServer**

Trong Master Server có chứa các thông số chính phục vụ cho việc khởi động và cấu hình trạng thái hoạt động của server master. Các thông số bao gồm : Địa chỉ IP của master, số hiệu cổng, địa chỉ database, số hiệu cổng của database, “username” của database, mật khẩu để truy nhập database, …. Để thuận tiện cho việc cài đặt và triển khai server, các thông số trạng thái của master server sẽ được lưu trong một file “config”, mỗi khi server khởi động nó sẽ lấy các thông số từ file config và tự động cấu hình các trạng thái. Khi cần triển khai server sang một môi trường mới chúng ta chỉ cần chỉnh sửa các thông số trong file “config”.

Ngoài các thông số trạng thái, trong MasterServer còn chứa các thành phần quan trọng khác:

* *dataEngine* : Là một đối tượng của lớp DataEngine phụ trách việc kết nối và thực hiện các thao tác với cơ sở dữ liệu.
* *logWriter* : Phụ trách việc ghi lại “log” của hệ thống.
* *socketServer* : Một đối tượng của lớp SocketServer phụ trách việc thiết lập kết nối giữa các client với master server.
* *users* : Một bảng băm lưu trữ các “user” đang đăng nhập trong hệ thống. Thông qua đó chúng ta có thể quản lý đươc hiện trạng của các “user” đang tồn tại trên hệ thống. Các “user” có thể là các thiết bị di động của người dùng đang kết nối, hoặc cũng có thể là các slave server.

**b. Các phương thức của MasterServer:**

|  |  |
| --- | --- |
| Void | *acceptConnection()*  Phương thức thực hiện lắng nghe kết nối từ client. Khi có một yêu cầu kết nối được gửi đến địa chỉ và cổng kết nối của master server thì sẽ tiến hành thiết lập kết nối giữa master server và client. |
| Void | *loadConfig()*  Phương thức thực hiện việc đọc file “config” và trả ra các thông số cấu hình của server. Hàm này được gọi khi khởi tạo MasterServer. |
| Void | *initLogger()*  Thiết lập và khởi tạo mô đun ghi “log” của server. |

Bảng 7. Các phương thức chính của lớp MasterServer.

**4.4.1.2 ConfigReader**

**a. Các thành phần chính của ConfigReader**

Như phần trên đã giới thiệu ConfigReader có nhiệm vụ đọc các thông tin cấu hình server từ file “config” từ đó thiết lập các thông số cho việc khởi chạy server. Hai thành phần quan trọng của ConfigReader là :

* *fileName* : Lưu trữ đường dẫn tới file “config”.
* *configTable* : Là một bảng băm lưu trữ các thông tin đọc được từ file “config”, server sẽ lấy ra các thông tin cần thiết từ bảng băm này.

**b. Các phương thức**

|  |  |
| --- | --- |
|  | *ConfigReader()*  Phương thức khởi tạo. Khởi tạo *configTable,* dựa vào *fileName* để đọc file config và lưu lại các thông số. Khi file config chưa tồn tại thì khởi tạo các giá trị config mặc định của hệ thống. |
| Void | *writeConfig()*  Thực hiện việc viết các thông tin cấu hình vào file config. |
| String | *getValuer(String key)*  Lấy thông số cấu hình của hệ thống từ *configTable* dựa vào giá trị “key” |
| Void | *setValuer(String key, String value)*  Lưu lại giá trị cấu hình của hệ thống vào *configTable*  với khóa “key” và giá trị “value” |

Bàng 8. Các phương thức của ConfigReader(MasterServer).

**4.4.1.3 LogWriter**

Đây là thành phần phụ trách nhiệm vụ ghi lại “log” của server. Với các thông tin được ghi lại trong file “log” người quản trị hệ thống có thể biết được toàn bộ tình trạng hoạt động của hệ thống trong suốt quá trình vận hành, từ đó phát hiện những bất thường hoặc lỗi phát sinh để tìm cách giải quyết cho phù hợp. Các thông số cần ghi lại của MasterServer bao gồm :

* Những yêu cầu kết nối tới master và được chấp nhận khởi tạo kết nối. Từ những thông tin này có thể nắm được số lượng “request” gửi tới master cũng như danh sách những user đã kết nối với master.
* Thông tin và trạng thái của các slave gửi về cho master. Thông tin và trạng thái của người chơi khi bước đầu đăng nhập vào hệ thống.
* Thông tin về những giao dịch mà người dùng sử dụng những dịch vụ của master (nạp tiền, kích hoạt tài khoản, .. ).
* Các lỗi gặp phải khi master vận hành.

LogWriter sử dụng bộ thư viện hỗ trợ ghi “log” miễn phí “log4j”, đây là bộ thư viện mã nguồn mở được sử dụng trong rất nhiều các hệ thống server và web để ghi lại “log” của hệ thống.

### 4.4.2 Mô đun quản lý kết nối

### Đây là thành phần phụ trách duy trì kết nối và trao đổi thông tin với client. Nhiệm vụ của mô đun này là lắng nghe các yêu cầu được gửi lên từ client, xử lý các yêu cầu đó và trả ra kết quả mà client mong muốn, các kết quả sau đó sẽ được gửi tới client.

### 4.4.2.1 MasterClient

### a. Các thành phần của MasterClient

### Khi client gửi yêu cầu kết nối lên master, nếu yêu cầu kết nối được chấp nhận master sẽ sinh ra một kênh kết nối riêng để phụ trách kết nối với client, MasterClient là thành phần chính quản lý kênh kết nối này. Từ đó ta có thể hiểu rằng một client sẽ tương ứng với một MasterClient, và các MasterClient đều được sinh ra và quản lý bởi master server, cũng vì vậy các client có thể liên lạc, truyền thông tin qua lại lẫn nhau, “tương tác” với nhau đúng với hình thức của một game online. Các thành phần chính của MasterClient gồm :

* *socket* : Là một đối tượng Socket được master server sinh ra khi khởi tạo kết nối với client, dựa vào “socket” được cung cấp này MasterClient sẽ trực tiếp thiết lập các phương thức trao đổi thông tin với client.
* *connectionHandler* : Một đối tượng ConnectionHandler có nhiệm vụ chính là duy trì việc đọc dữ liệu từ client gửi lên, gửi dữ liệu lại cho client và kiểm tra tình trạng của kết nối. ConnectionHandler sẽ được đề cập kĩ hơn ở phần sau.
* *server* : Là đối tượng MasterServer trực tiếp quản lý MasterClient.

**b. Các phương thức của MasterClient**

|  |  |
| --- | --- |
|  | *MasterClient(MasterServer server, Socket socket)*  Phương thức khởi tạo MasterClient, tham số truyền vào bao gồm server và socket để khởi tạo các giá trị như trên đã trình bày. |
| void | *icomingMessage(String message)*  Phương thức phụ trách xử lý thông tin được client gửi tới. Gói tin được gửi lên dưới dạng một chuỗi kí tự, dựa vào những quy tắc về gói tin có từ trước, MasterClient sẽ bóc tách và xử lý các yêu cầu. |
| void | *processRequest(String data, short appID, int itemID)*  Phương thức phụ trách gửi thông tin về client. Thông tin được chứa trong một chuỗi “data”, *appID* là mã yêu cầu để client nhận biết, *itemID* là mã để phân loại gói tin |
| Void | *dispose() :* Hàm xử lý khi ngắt kết nối với client. |

Bảng 9: Các phương thức của MasterClient.

**4.4.2.2 ConnectionHandler**

**a. Các thành phần chính của ConnectionHandler**

Khi client thực hiện thiết lập kết nối với master server một kênh kết nối được thành lập. Nếu như MasterClient chịu trách nhiệm chính trong việc quản lý và xử lý thông tin thì ConnectionHandler phụ trách việc “vận chuyển” thông tin trao đổi giữa client và master server. Các thành phần của ConnectionHandler bao gồm:

* *client :* Là đối tượng MasterClient được khởi tạo khi hình thành kết nối giữa client và server.
* *dispatcher :* Là đối tượng của lớp Dispatcher có nhiệm vụ truyền thông tin từ server tới client.
* *reader :* Là đối tượng của lớp ReaderThread có nhiệm vụ tiếp nhận và phần loại thông tin được gửi lên từ client.

**b. Các phương thức của ConnectionHandler**

|  |  |
| --- | --- |
| Void | *init()*  Phương thức có nhiệm vụ khởi tạo các yếu tố cần thiết cho việc kết nối, khởi chạy hai Thread dispatcher và reader. |
| Void | *queueMessage(String message)*  Đẩy gói tin cần truyển tới client vào queue để chờ *dispatcher* gửi đi. |
| Void | *shutdown()*  Đóng kết nối |

Bảng 10: Các phương thức của ConnectionHandler.

**4.4.2.3 Dispatcher**

**a. Các thành phần của Dispatcher**

Dispatcher có nhiệm vụ gửi thông tin tới client ngay khi có yêu cầu.Các gói tin cần được gửi đi sẽ được ConnectionHandler đẩy vào *“queue”,* các gói tin sẽ được gửi theo thứ tự tới khi trong “queue” không còn gói tin nào khác.

**b. Các phương thức của Dispatcher**

|  |  |
| --- | --- |
| Void | *run()*  Dispatcher là 1 Thread chạy độc lập, hàm *run()* chính là hàm chạy của Thread thực thi tất cả các tác vụ của Dispatcher. |
| Void | *queueMessage(String message)*  Hàm thực thi đẩy gói tin vào queue để chờ xử lý |
| Void | *halt()*  Hàm thực hiện đóng Dispatcher và ngắt các kết nối. |

Bảng 11. Các phương thức của Dispatcher trong Master Server.

**4.4.2.4 ReaderThread**

**a. Các thành phần của ReaderThread**

Giống với Dispatcher, ReaderThread cũng được thừa kế từ Thread và thực thi một cách độc lập. Nhiệm vụ của ReaderThread là đọc các gói tin từ client gửi lên, bóc tách các gói tin, phân loại chúng rồi đẩy vào *“queue”* để chờ MasterServer xử lý.

**b. Các phương thức của ReaderThread**

|  |  |
| --- | --- |
| Void | *run()*  ReaderThread là 1 Thread chạy độc lập, hàm *run()* chính là hàm chạy của Thread thực thi tất cả các tác vụ của ReaderThread. |
| Void | *halt()*  Hàm thực hiện đóng ReaderThread và ngắt các kết nối. |

Bảng 12: Các phương thức của ReaderThread trong MasterServer.

### 4.4.3 Mô đun thao tác và quản lý cơ sở dữ liệu

Đây là mô đun được thiết kế độc lập có nhiệm vụ chính là quản lý và thực hiện các tác vụ với cơ sở dữ liệu của hệ thống ( truy vấn thông tin, thêm, sửa, xóa, … ). Cơ sở dữ liệu của hệ thống em xây dựng dựa trên nền tảng MySQL, các kết nối với cơ sở dữ liệu có sử dụng tới thư viện JDBC và JdbcSpringFrameWork của Java. Khi Master Server muốn thực hiện một thao tác tới cơ sở dữ liệu, nó sẽ thực hiện qua việc request tới một thể hiện của một giao diện *IdataEngine.* Trong *IdataEngine* có chứa đầy đủ các phương thức lấy dữ liệu cần thiết của Master Server. Mô đun được thiết kế với 3 thành phần chính là : DataEngine, TableDao, Table

* Table: Là thể hiện của các bảng trong cơ sở dữ liệu dưới dạng hướng đối tượng. Mỗi bảng trong cơ sở dữ liệu sẽ được tương ứng với một lớp. Tên của lớp là tên của bảng, các thuộc tính của lớp chính là các trường của bảng.
* TableDao : Thực hiện tất cả các câu lệnh truy vấn SQL tới cơ sở dữ liệu đồng thời thực hiện “map” những dữ liệu truy vấn được về các đối tượng Table.
* DataEngine : Được thực hiện giao diện *IdataEngine*, có đầy đủ các phương thức thực hiện các yêu cầu của server với cơ sở dữ liệu ( Thực hiện thông qua đối tượng TableDao). Trong DataEngine cũng có chứa các thông số của cơ sở dữ liệu hệ thống : Địa chỉ máy chủ đặt server, số hiệu cổng, tài khoản, mật khẩu, tên cơ sở dữ liệu để khởi tạo kết nối tới cơ sở dữ liệu.

## 4.5 Slave Server

### 4.5.1 Mô đun quản lý server

Mô đun gồm các thành phần phụ trách việc quản lý hoạt động, thiết lập cấu hình và các trạng thái của slave. Mô đun này cũng phụ trách việc thiết lập kết nối với client khi có yêu cầu kết nối đồng thời quản lý trạng thái của các “user” đang hoạt động trên server.

**4.5.1.1 Slave Server**

**a. Các thành phần chính của SlaveServer**

Trong SlaveServer có chứa các thông số chính phục vụ cho việc khởi động và cấu hình trạng thái hoạt động của . Các thông số bao gồm : Địa chỉ IP của slave, số hiệu cổng, địa chỉ IP của slave, số hiệu cổng của slave, tên slave, địa chỉ database, số hiệu cổng của database, “username” của database, mật khẩu để truy nhập database, …. Để thuận tiện cho việc cài đặt và triển khai server, các thông số trạng thái của slave server sẽ được lưu trong một file “config”, mỗi khi slave khởi động nó sẽ lấy các thông số từ file config và tự động cấu hình các trạng thái. Khi cần triển khai server sang một môi trường mới chúng ta chỉ cần chỉnh sửa các thông số trong file “config”.

Ngoài các thông số trạng thái, trong SlaveServer còn chứa các thành phần quan trọng khác:

* *dataEngine* : Là một đối tượng của lớp DataEngine phụ trách việc kết nối và thực hiện các thao tác với cơ sở dữ liệu. Việc thao tác với cơ sở dữ liệu hoàn toàn do dataEngine phụ trách. Theo đó khi có bất kì sự thay đổi nào ở cơ sở dữ liệu chúng ta chỉ cần thay thế thành phần dataEngine mà không làm ảnh hưởng tới các thành phần khác của slave.
* *logWriter* : Phụ trách việc ghi lại “log” của hệ thống. Log cần ghi lại có thể gồm những cập nhật của người chơi, những giao dịch người chơi thực hiện trong game, những lỗi mà hệ thống gặp phải trong quá trình vận hành.
* *socketServer* : Một đối tượng của lớp SocketServer phụ trách việc thiết lập kết nối giữa các client với master server. Vì slave được xây dựng trên nền tảng ngôn ngữ Java nên việc khởi tạo kết nối sẽ do lớp SocketServer, được hỗ trợ trong thư viện của Java, phụ trách.
* *users* : Một bảng băm lưu trữ các “user” đang đăng nhập trong hệ thống. Thông qua đó chúng ta có thể quản lý đươc hiện trạng của các “user” đang tồn tại trên hệ thống.



Hình 29. Biểu đồ lớp của Slave Server.

Ngoài các thành phần trên Slave Server còn có thêm hai thành phần khác là SessionManager để quản lý các phiên làm việc hiện đang thực hiện trên server và MasterPinger có nhiệm vụ đảm bảo kết nối giữa Slave Server và Master Server. Hai thành phần này sẽ được giới thiệu chi tiết hơn ở phần sau.

**b. Các phương thức của SlaveServer**

|  |  |
| --- | --- |
| Void | *acceptConnection()*  Phương thức thực hiện lắng nghe kết nối từ client. Khi có một yêu cầu kết nối được gửi đến địa chỉ và cổng kết nối của master server thì sẽ tiến hành thiết lập kết nối giữa master server và client. |
| Void | *loadConfig()*  Phương thức thực hiện việc đọc file “config” và trả ra các thông số cấu hình của server. Hàm này được gọi khi khởi tạo MasterServer. |
| Void | *initLogger()*  Thiết lập và khởi tạo mô đun ghi “log” của server. |

Bảng 13: Các phương thức của SlaveServer

**4.5.1.2 ConfigReader**

**a. Các thành phần chính của ConfigReader**

Như phần trên đã giới thiệu ConfigReader có nhiệm vụ đọc các thông tin cấu hình server từ file “config” từ đó thiết lập các thông số cho việc khởi chạy server. Hai thành phần quan trọng của ConfigReader là :

* *fileName* : Lưu trữ đường dẫn tới file “config”.
* *configTable* : Là một bảng băm lưu trữ các thông tin đọc được từ file “config”, server sẽ lấy ra các thông tin cần thiết từ bảng băm này.

**b. Các phương thức**

|  |  |
| --- | --- |
|  | *ConfigReader()*  Phương thức khởi tạo. Khởi tạo *configTable,* dựa vào *fileName* để đọc file config và lưu lại các thông số. Khi file config chưa tồn tại thì khởi tạo các giá trị config mặc định của hệ thống. |
| Void | *writeConfig()*  Thực hiện việc viết các thông tin cấu hình vào file config. |
| String | *getValuer(String key)*  Lấy thông số cấu hình của hệ thống từ *configTable* dựa vào giá trị “key” |
| Void | *setValuer(String key, String value)*  Lưu lại giá trị cấu hình của hệ thống vào *configTable*  với khóa “key” và giá trị “value” |

Bảng 14. Các phương thức của ConfigReader(SlaveServer).

**4.5.1.3 LogWriter**

Đây là thành phần phụ trách nhiệm vụ ghi lại “log” của server. Với các thông tin được ghi lại trong file “log” người quản trị hệ thống có thể biết được toàn bộ tình trạng hoạt động của hệ thống trong suốt quá trình vận hành, từ đó phát hiện những bất thường hoặc lỗi phát sinh để tìm cách giải quyết cho phù hợp. Các thông số cần ghi lại của SlaveServer bao gồm :

* Những yêu cầu kết nối tới slave và được chấp nhận khởi tạo kết nối. Từ những thông tin này có thể nắm được số lượng “request” gửi tới slave cũng như danh sách những user đã kết nối với slave.
* Thông tin và trạng thái của người chơi khi bước đầu đăng nhập vào hệ thống.
* Thông tin về những giao dịch mà người dùng sử dụng những dịch vụ của slave (nạp tiền, kích hoạt tài khoản, .. ).
* Các lỗi gặp phải khi slave vận hành.

LogWriter sử dụng bộ thư viện hỗ trợ ghi “log” miễn phí “log4j”, đây là bộ thư viện mã nguồn mở được sử dụng trong rất nhiều các hệ thống server và web để ghi lại “log” của hệ thống.

**4.5.1.4 MasterPinger**

Đây là thành phần chính phụ trách việc thực hiện kết nối giữa slave và master. Giữa slave mà master có thống nhất một phương thức kết nối, theo đó trong một khoảng thời gian nhất định slave sẽ “ping” một gói tin chứa thông tin và trạng thái hoạt động của mình tới master. MasterPinger thực hiện thực hiện kết nối tới master và gửi các thông tin về hiện trạng của slave tới master. Trong trường hợp kết nối không thành công hoặc gói tin gửi đi và không có thông tin gửi từ master thì có nghĩa là kết nối giữa slave và master hoặc master server đã gặp sự cố. Khi tình huống này xảy ra slave sẽ chuyển sang chế độ sẵn sàng lắng nghe kết nối trực tiếp từ client mà không thông qua master, khi kết nối được thiết lập lại slave sẽ trở về trạng thái bình thường.

Về phía master, khi nhận được gói tin từ MasterPinger của slave thì gửi thông tin hồi đáp lại, đồng thời cập nhật trạng thái hiện tại của slave. Nếu trong một khoảng thời gian định sẵn mà không nhận được thông tin từ slave thì có thể kết nối của slave đã gặp sự cố, master sẽ loại slave khỏi danh sách các slave đáp ứng, khi người dùng đăng nhập vào hệ thống sẽ được thông báo để không truy cập vào slave bị lỗi.

### 4.5.2 Mô đun quản lý kết nối

Đây là thành phần phụ trách duy trì kết nối và trao đổi thông tin với client. Nhiệm vụ của mô đun này là lắng nghe các yêu cầu được gửi lên từ client, xử lý các yêu cầu đó và trả ra kết quả mà client mong muốn, các kết quả sau đó sẽ được gửi tới client.

### 4.5.2.1 SlaveClient

### a. Các thành phần của SlaveClient

### Khi client gửi yêu cầu kết nối lên slave, nếu yêu cầu kết nối được chấp nhận slave sẽ sinh ra một kênh kết nối riêng để phụ trách kết nối với client, SlaveClient là thành phần chính quản lý kênh kết nối này. Từ đó ta có thể hiểu rằng một client sẽ tương ứng với một SlaveClient, và các SlaveClient đều được sinh ra và quản lý bởi slave server, cũng vì vậy các client có thể liên lạc, truyền thông tin qua lại lẫn nhau, “tương tác” với nhau đúng với hình thức của một game online. Các thành phần chính của SlaveClient gồm :

* *socket* : Là một đối tượng Socket được slave server sinh ra khi khởi tạo kết nối với client, dựa vào “socket” được cung cấp này SlaveClient sẽ trực tiếp thiết lập các phương thức trao đổi thông tin với client.
* *connectionHandler* : Một đối tượng ConnectionHandler có nhiệm vụ chính là duy trì việc đọc dữ liệu từ client gửi lên, gửi dữ liệu lại cho client và kiểm tra tình trạng của kết nối. ConnectionHandler sẽ được đề cập kĩ hơn ở phần sau.
* *server* : Là đối tượng SlaveServer trực tiếp quản lý SlaveClient.

**b. Các phương thức của SlaveClient**

|  |  |
| --- | --- |
|  | *SlaveClient(SlaveServer server, Socket socket)*  Phương thức khởi tạo SlaveClient, tham số truyền vào bao gồm server và socket để khởi tạo các giá trị như trên đã trình bày. |
| Void | *icomingMessage(String message)*  Phương thức phụ trách xử lý thông tin được client gửi tới. Gói tin được gửi lên dưới dạng một chuỗi kí tự, dựa vào những quy tắc về gói tin có từ trước, SlaveClient sẽ bóc tách và xử lý các yêu cầu. |
| Void | *processRequest(String data, short appID, int itemID)*  Phương thức phụ trách gửi thông tin về client. Thông tin được chứa trong một chuỗi “data”, *appID* là mã yêu cầu để client nhận biết, *itemID* là mã để phân loại gói tin |
| Void | *dispose() :* Hàm xử lý khi ngắt kết nối với client. |

Bảng 15: Các phương thức của SlaveClient.

**4.5.2.2 ConnectionHandler**

**a. Các thành phần chính của ConnectionHandler**

Khi client thực hiện thiết lập kết nối với master server một kênh kết nối được thành lập. Nếu như SlaveClient chịu trách nhiệm chính trong việc quản lý và xử lý thông tin thì ConnectionHandler phụ trách việc “vận chuyển” thông tin trao đổi giữa client và slave server. Các thành phần của ConnectionHandler bao gồm:

* *client :* Là đối tượng SlaveClient được khởi tạo khi hình thành kết nối giữa client và server.
* *dispatcher :* Là đối tượng của lớp Dispatcher có nhiệm vụ truyền thông tin từ server tới client.
* *reader :* Là đối tượng của lớp ReaderThread có nhiệm vụ tiếp nhận và phần loại thông tin được gửi lên từ client.

**b. Các phương thức của ConnectionHandler**

|  |  |
| --- | --- |
| Void | *init()*  Phương thức có nhiệm vụ khởi tạo các yếu tố cần thiết cho việc kết nối, khởi chạy hai Thread dispatcher và reader. |
| Void | *queueMessage(String message)*  Đẩy gói tin cần truyển tới client vào queue để chờ *dispatcher* gửi đi. |
| Void | *shutdown()*  Đóng kết nối |

Bảng 16: Các phương thức của ConnectionHandler trong SlaveServer.

**4.5.2.3 Dispatcher**

**a. Các thành phần của Dispatcher**

Dispatcher được thừa kế từ Thread có nhiệm vụ gửi thông tin tới client ngay khi có yêu cầu. Các thông tin trước khi gửi đi, Dispatcher sẽ tiến hành định dạng lại cho đúng với khuôn dạng gói tin đã được quy định với client. Các gói tin cần được gửi đi sẽ được ConnectionHandler đẩy vào *“queue”,* các gói tin sẽ được gửi theo thứ tự tới khi trong “queue” không còn gói tin nào khác.

**b. Các phương thức của Dispatcher**

|  |  |
| --- | --- |
| Void | *run()*  Dispatcher là 1 Thread chạy độc lập, hàm *run()* chính là hàm chạy của Thread thực thi tất cả các tác vụ của Dispatcher. |
| Void | *queueMessage(String message)*  Hàm thực thi đẩy gói tin vào queue để chờ xử lý |
| Void | *halt()*  Hàm thực hiện đóng Dispatcher và ngắt các kết nối. |

Bảng 17. Các phương thức của Dispatcher trong Slave Server.

**4.5.2.4 ReaderThread**

Giống với Dispatcher, ReaderThread cũng được thừa kế từ Thread và thực thi một cách độc lập. Nhiệm vụ của ReaderThread là đọc các gói tin từ client gửi lên, bóc tách các gói tin, phân loại chúng rồi đẩy vào *“queue”* để chờ SlaveServer xử lý.

**Các phương thức của ReaderThread**

|  |  |
| --- | --- |
| Void | *run()*  ReaderThread là 1 Thread chạy độc lập, hàm *run()* chính là hàm chạy của Thread thực thi tất cả các tác vụ của ReaderThread. |
| Void | *halt()*  Hàm thực hiện đóng ReaderThread và ngắt các kết nối. |

Bảng 18: Các phương thức của ReaderThread trong SlaveServer.

### 4.5.3 Mô đun thao tác và quản lý cơ sở dữ liệu

Đây là mô đun được thiết kế độc lập có nhiệm vụ chính là quản lý và thực hiện các tác vụ với cơ sở dữ liệu của hệ thống ( truy vấn thông tin, thêm, sửa, xóa, … ). Cơ sở dữ liệu của hệ thống em xây dựng dựa trên nền tảng MySQL, các kết nối với cơ sở dữ liệu có sử dụng tới thư viện JDBC và JdbcSpringFrameWork của Java. Khi Slave Server muốn thực hiện một thao tác tới cơ sở dữ liệu, nó sẽ thực hiện qua việc request tới một thể hiện của một giao diện *IdataEngine.* Trong *IdataEngine* có chứa đầy đủ các phương thức lấy dữ liệu cần thiết của Slave Server. Mô đun được thiết kế với 3 thành phần chính là : DataEngine, TableDao, Table

* Table: Là thể hiện của các bảng trong cơ sở dữ liệu dưới dạng hướng đối tượng. Mỗi bảng trong cơ sở dữ liệu sẽ được tương ứng với một lớp. Tên của lớp là tên của bảng, các thuộc tính của lớp chính là các trường của bảng.
* TableDao : Thực hiện tất cả các câu lệnh truy vấn SQL tới cơ sở dữ liệu đồng thời thực hiện “map” những dữ liệu truy vấn được về các đối tượng Table.
* DataEngine : Được thực hiện giao diện *IdataEngine*, có đầy đủ các phương thức thực hiện các yêu cầu của server với cơ sở dữ liệu ( Thực hiện thông qua đối tượng TableDao). Trong DataEngine cũng có chứa các thông số của cơ sở dữ liệu hệ thống : Địa chỉ máy chủ đặt server, số hiệu cổng, tài khoản, mật khẩu, tên cơ sở dữ liệu để khởi tạo kết nối tới cơ sở dữ liệu.

## 4.6 Client

Client chính là game chạy trên các thiết bị di động của người sử dụng. Nhiệm vụ của client là gửi yêu cầu lên server, nhận các dữ liệu đã được server xử lý và hiển thị tới người sử dụng. Vì tốc độ xử lý và kết nối mạng của các thiết bị di động chắc chắn không thể nhanh như server vì vậy cần có cơ chế xử lý để tránh tình trạng thất lạc và mất gói tin. Bên cạnh đó các thông tin được gửi về cũng cần được hiển thị một cách nhanh nhất tới người sử dụng, vì một trong những yếu tố rất quan trọng của game online là sự chính xác về thời gian.



Hình 30. Biểu đồ lớp của client.

### 4.6.1 GameClient

Đây chính là thành phần chính phụ trách quản lý các kết nối tới server của client. GameClient chịu trách nhiệm gửi yêu cầu kết nối tới server và thiết lập kênh kết nối với server khi yêu cầu kết nối được chấp nhận. Trong GameClient có lưu trữ các thông tin cần thiết để thiết lập kết nối : Địa chỉ của server và số hiệu cổng của server. Khi thực hiện kết nối GameClient sẽ mở kết nối tới server, khi kết nối được chấp nhận một đối tượng *socket* sẽ được tạo ra và sử dụng để khởi tạo các kết nối với server.

**\* Các phương thức của GameClient**

|  |  |
| --- | --- |
| Void | *connect()*  Thực hiện việc kết nối tới server |
| Void | *isConnected()*  Kiểm tra xem hiện tại client có còn giữ kết nối với server hay không. |
| Void | *close()*  Đóng kết nối với server |
| Void | *sendMessage(String message)*  Gửi yêu cầu tới server thông qua gói tin *message.* |

Bảng 19: Các phương thức của GameClient.

### 4.6.2 ConnectionHandler

Giống với server phía client cũng có một thành phần ConnectionHandler để quản lý việc “vận chuyển” thông tin qua lại giữa client và server. Trong ConnectionHandler cũng có hai luồng đọc và ghi dữ liệu (*reader* và *writer*).

**\* Các phương thức của ConnectionHandler**

|  |  |
| --- | --- |
| Void | *sendMessage(String message)*  Gửi thông điệp lên server thông qua WriterThread |
| Void | *close()*  Đóng kết nối với server |
| Void | *init()*  Khởi tạo luồng đọc và ghi dữ liệu theo đối tượng socket được tạo ra khi thiết lập kết nối ở GameClient. |

Bảng 20: Các phương thức của ConnectionHandler ở client.

### 4.6.3. ReaderThread

ReaderThread cũng được thừa kết từ một giao diện Runnable và thực thi một cách độc lập. Nhiệm vụ của ReaderThread là đọc các gói tin từ server gửi xuống, bóc tách các gói tin, phân loại chúng rồi đẩy vào *“queue”* để chờ xử lý.

**\* Các phương thức của ReaderThread**

|  |  |
| --- | --- |
| Void | *run()*  Hàm thực thi Thread, thực hiện việc liên tục bóc tách các gói tin được gửi về từ server thành các DataPackage |
| Void | *processData()*  Xử lý các thông tin từ server gửi về |
| Void | *close()*  Đóng Thread đọc |

Bảng 21: Các phương thử của ReaderThread ở client.

### 4.6.4 WriterThread

WriterThread được thừa kế từ giao diện Runnable có nhiệm vụ gửi thông tin tới server ngay khi có yêu cầu. Các thông tin trước khi gửi đi, WriterThread sẽ tiến hành định dạng lại cho đúng với khuôn dạng gói tin đã được quy định với server. Các gói tin cần được gửi đi sẽ được ConnectionHandler đẩy vào *“queue”,* các gói tin sẽ được gửi theo thứ tự tới khi trong “queue” không còn gói tin nào khác.

**\* Các phương thức của WriterThread**

|  |  |
| --- | --- |
| Void | *run()*  Hàm thực thi của Thread |
| Void | *queueMessage(String message)*  Đưa message vào queue để chờ được gửi lên server |
| Void | *sendData()*  Gửi dữ liệu lên server |
| Void | *close()*  Đóng luồng ghi |

Bảng 22: Các phương thức của WriterThread ở client.

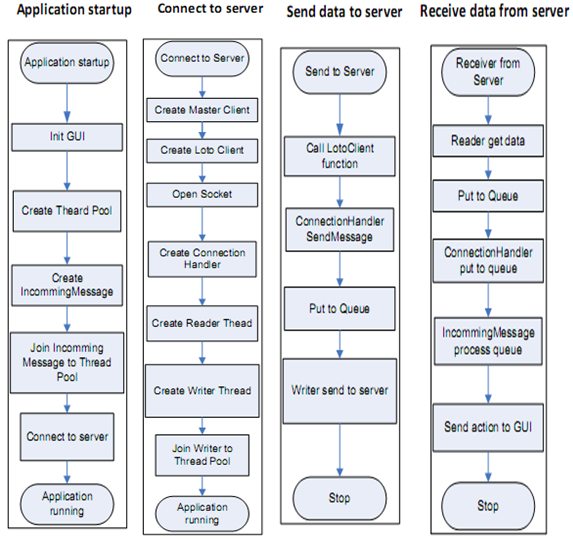
### 4.6.5 Định dạng gói tin

Dữ liệu được server truyền về client tùy theo yêu cầu của client sẽ có thể là các dữ liệu chuỗi kí tự, hình ảnh, âm thanh, … client cần phải tổ chức phân loại các gói tin để xử lý. Việc phân loại gói tin được thực hiện trong ReaderThread, thông tin được truyền đi bởi DataOutputStream qua socket ở dưới dạng byte, ReaderThread sẽ dựa vào các bit đánh dấu được quy định trước để bóc tách gói tin. Trước đó chúng ta cần phải đưa ra một định dạng gói tin để ReaderThread phân loại. Chúng ta định nghĩa một lớp DataPackage là một lớp trừu tượng làm chuẩn cho việc định dạng các gói tin gửi về. DataPackage có các thành phần chính là : appID, itemID và type.

* appID : Là thông số chứa mã của yêu cầu. Mỗi yêu cầu của client gửi lên sẽ được server gửi về thông tin tương ứng và *appID* chính là thông số đánh dấu gói tin gửi về là của yêu cầu nào để client có cách thức xử lý phù hợp.
* itemID : Tương ứng với một yêu cầu của client tùy vào trạng thái hiện giờ server sẽ gửi về các gói tin với trạng thái khác nhau, *itemID* chính là thành phần đánh dấu trạng thái này.
* type : Gói tin được gửi tới client có thể ở rất nhiều định dạng khác nhau như : Chuỗi kí tự, ảnh, âm thanh, … biến type được dùng để đánh dấu loại gói tin gửi về từ đó client sẽ đưa ra các phương thức tái tạo gói tin phù hợp.

Từ DataPackage sẽ có các gói riêng biệt thừa kế tùy theo loại gói tin như ContentPackage là gói chứa nội dung là chuỗi kí tự, ImagePackage là gói chứa nội dung là hình ảnh. Do đặc trưng của kết nối trên các thiết bị di động nên các gói tin truyền về không được quá lớn. Với các gói dữ liệu có kích thước quá lớn ta phải tiến hành chia các gói tin thành các đoạn và sẽ tiên hành tổ hợp sau khi đã nhận đủ các gói tin. Để tránh tình trạng mất mát gói tin, mỗi gói tin gửi đi đều có 1 thông số *segment* để đánh dấu, nếu thông số *segment = -1* thì gói tin đó là gói tin cuối cùng và việc tổ hợp gói tin được hoàn tất.

### 4.6.6 Mô hình cấu trúc các mô đun của client



Hình 31: Sơ đồ cấu trúc các mô đun của client.

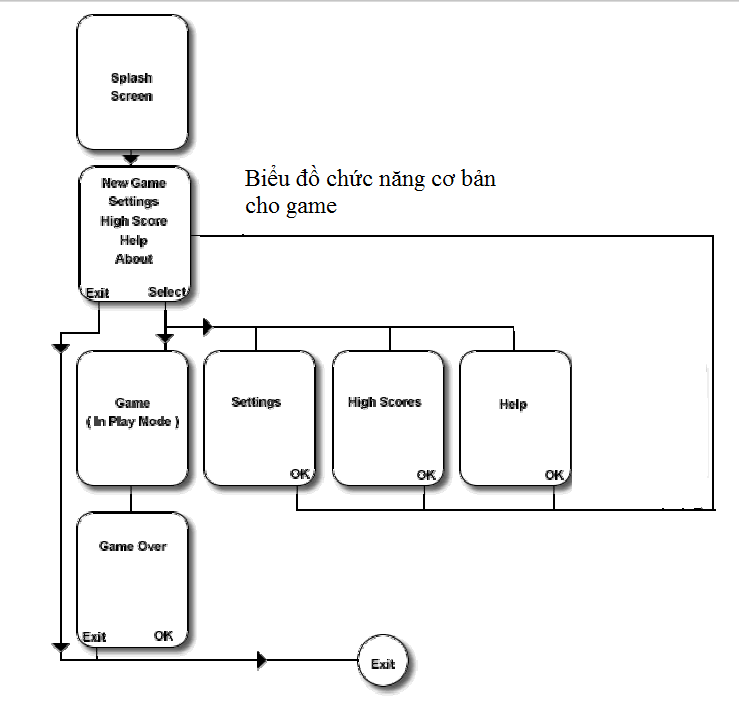
# CHƯƠNG 5. CÀI ĐẶT VÀ TRIỂN KHAI HỆ THỐNG

## 5. 1 Giao diện game

### 5.1.1 Giao diện cơ bản của game trên di động

Một game trên di động thường được xây dựng với các giao diện gồm

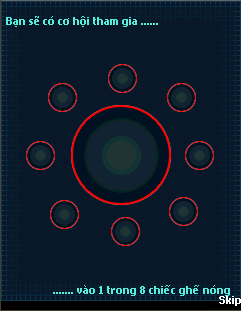
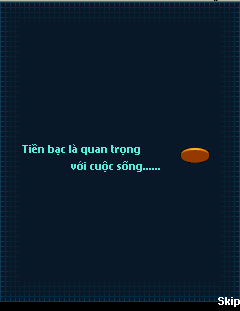
* Splash Screen : Màn hình hiện ra khi bắt đầu game, thường là màn hình giới thiệu về game và tác giả của game.
* Menu : Menu chính với những chức năng chính của game để người chơi lựa chọn.
* Game Screen: Là giao diện chính để chơi game.
* Các giao diện trợ giúp khác như : Cài đặt, điểm cao, hướng dẫn…
* Màn hình khi kết thúc game.

****

Hình 32: Các giao diện cơ bản của game trên di động.

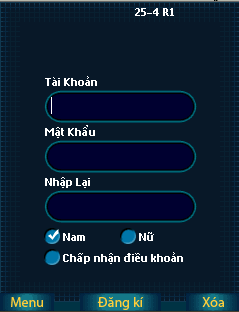
### 5.1.2 Giao diện game Đấu trường trí tuệ

**5.1.2.1. Giao diện giới thiệu game**

****

Hình 33: Giao diện giới thiệu game.

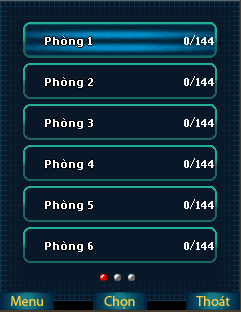
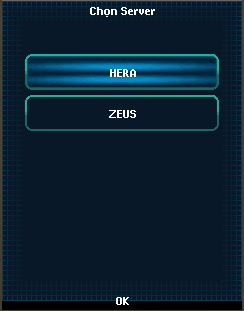
**5.1.2.2. Giao diện đăng ký, Đăng nhập**

****

Hình 34: Giao diện đăng ký, đăng nhập.

Với giao diện đăng ký, người sử dụng nhập tên tài khoản, mật khẩu và nhắc lại mật khẩu của mình vào các ô nhập. Lựa chọn giới tính đăng ký và đánh dấu vào ô chấp nhận điều khoản để đăng ký. Sau khi đăng ký thành công người chơi sẽ được chuyển đến giao diện đăng nhập, người chơi nhập tài khoản đã được đăng ký và mật khẩu để đăng nhập vào hệ thống.

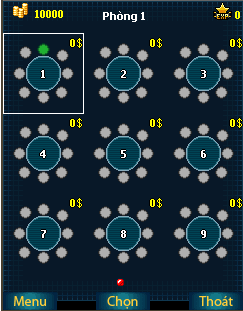
**5.1.2.3. Giao diện chọn server và chọn phòng**

****

Hình 35: Giao diện chọn server và chọn phòng.

Sau khi đăng nhập thành công vào hệ thống, người chơi sẽ lựa chọn server mà mình muốn tham gia chơi, danh sách các server này sẽ được hệ thống (master server) gửi về client. Lựa chọn server thành công, người chơi sẽ đến giao diện chọn phòng để tham gia chơi.

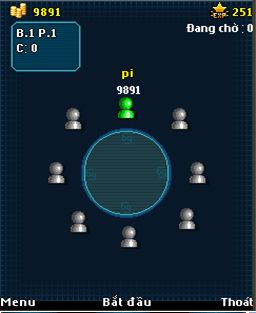
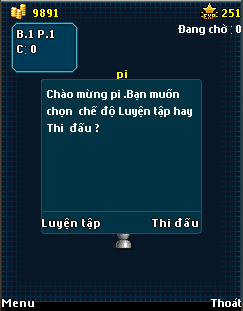
**5.1.2.4. Giao diện chọn bàn chơi**

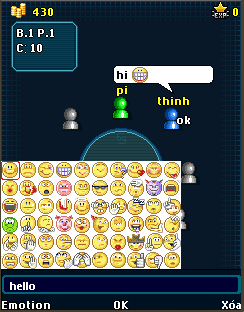
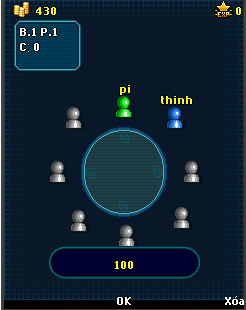
****

Hình 36: Giao diện chọn bàn chơi.

Trong bàn chơi có danh sách các bàn để người chơi có thể lựa chọn vào chơi. Mỗi bàn có 8 vị trí cho người chơi, tùy vào màu sắc của các vị trí người chơi sẽ biết được trạng thái của bàn. Số tiền cược và trạng thái của mỗi bàn cũng được hiển thị tới người chơi. Khi đang tìm bàn người chơi cũng có thể nhận được những lời mời tham gia chơi từ chủ các bàn trong phòng.

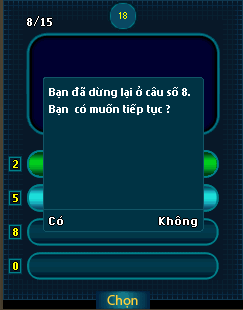
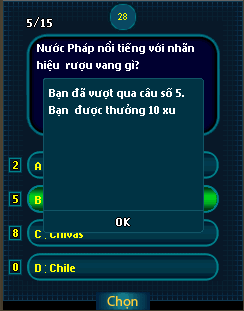
**5.1.2.5. Giao diện trong bàn chơi**

****

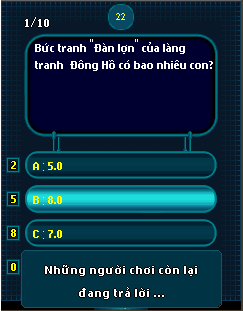
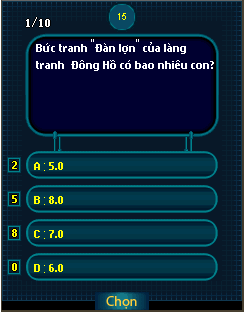
****

Hình 37: Giao diện trong bàn chơi.

**5.1.2.6. Giao diện trả lời câu hỏi**

****

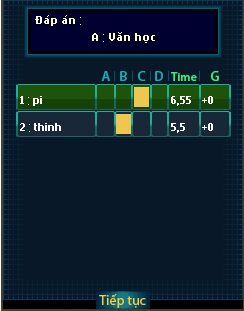
Hình 38a: Giao diện trả lời câu hỏi khi luyện tập.

**

Hình 38b: Giao diện trả lời câu hỏi khi thi đấu.

Mỗi câu hỏi có nội dung câu hỏi và 4 phương án trả lời, người chơi có thể dùng phím mũi tên để chọn câu hỏi hoặc dùng các phím tắt hỗ trợ được hiển thị trên giao diện. Giao diện cũng hỗ trợ các điện thoại cảm ứng nên người chơi có thể chạm vào các đáp án muốn chọn để trả lời. Giao diện của luyện tập và thi đấu chỉ có đôi chút khác biệt ở các thông báo đưa ra trên màn hình (Như hình 37a và 37b).

**5.1.2.7. Giao diện kết quả**

****

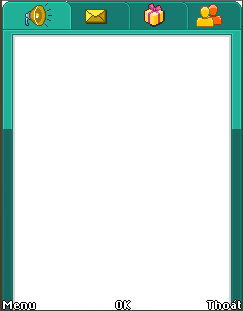
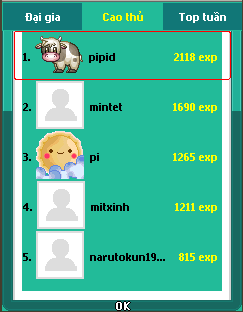
Hình 39: Giao diện kết quả.

Giao diện kết quả được hiển thị sau mỗi lượt chơi, trên giao diện hiển thị đáp án mà các người chơi lựa chọn, thời gian trả lời của mỗi người chơi và điểm số mà người chơi đạt được sau mỗi câu hỏi.

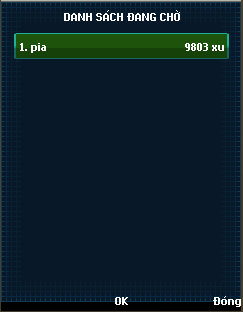
**5.1.2.8. Các giao diện khác trong game**

****

Hình 40a: Giao diện cửa hàng và Giao diện thông tin cá nhân

**

Hình 40b: Giao diện Top người chơi và Hộp thư.



Hình 40c: Giao diện nạp tiền và xem danh sách người đang chờ.

## 5.2 Triển khai hệ thống

### 5.2.1 Môi trường và công cụ sử dụng

Các thành phần của hệ thống được xây dựng dựa trên nền tảng và các công cụ:

* Server của hệ thống được xây dựng trên nền tảng Java SE 6, có sử dụng các thư viện của log4j và SpringFrameworks.
* Cơ sở dữ liệu được xây dựng bằng MySQL, kết nối với cơ sở dữ liệu có sử dụng thư viện Jdbc Driver của Java.
* Client được xây dựng trên 2 nền tảng là J2ME và Android.
* Máy chủ để chạy server và các dịch vụ được cài đặt Window Server 2003 có cài đặt môi trường Java, MySQL Server và Apache Server.

### 5.2.2 Cài đặt và cấu hình hệ thống

Trước hết chúng ta cần cài đặt cơ sở dữ liệu của hệ thống. Cài đặt MySQL Server và xây dựng cơ sở dữ liệu như đã thiết kế. Khi cài đặt MySQL Server cần đặc biệt chú ý đến 3 thông số : Tài khoản, mật khẩu và tên Database.

Sau khi đã cài đặt cơ sở dữ liệu cho hệ thống. Chúng ta tiến hành sửa các thông tin cấu hình của server. File config được đặt cùng thư mục với file thực thi của server. Với Master Server chúng ta chỉnh các thông số cấu hình như sau:

* LOGFILE\_LOCATION: Lưu địa chỉ của thư mục chứa các file log của master server.
* MASTER\_IPADDR: Địa chỉ IP của master.
* MASTER\_PORT: Số hiệu cổng của master.
* MASTER\_DB\_PORT: Số hiệu cổng kết nối của Database.
* MASTER\_DB\_SID: Tên Database.
* MASTER\_DB\_USERNAME: Tài khoản để đăng nhập vào MySQL Server.
* MASTER\_DB\_PASSWORD: Mật khẩu đăng nhập của MySQL Server.

Với Slave Server file config cũng được đặt cùng một thư mục với file thực thi của Slave Server, các thông số cấu hình được chỉnh sửa như sau:

* LOGFILE\_LOCATION: Lưu địa chỉ của thư mục chứa các file log của slave server.
* DB\_PORT: Số hiệu cổng kết nối của Database.
* DB\_SID: Tên Database.
* DB\_USERNAME: Tài khoản để đăng nhập vào MySQL Server.
* DB\_PASSWORD: Mật khẩu đăng nhập của MySQL Server.
* MASTER\_IPADDR: Địa chỉ IP của master.
* MASTER\_PORT: Số hiệu cổng của master.
* SLAVE\_IPADDR: Địa chỉ IP của slave.
* SLAVE\_PORT: Số hiệu cổng của slave.
* SLAVE\_ID: Id của slave.
* SLAVE\_NAME: Tên của slave, được dùng để phân biệt với các slave khác.

Sau khi cài đặt các thông số cho cả master và slave, chúng ta khởi chạy master và các slave. Khi khởi chạy chúng sẽ tự sinh các thư mục chứa log đã được chỉ định trong file config, slave sẽ dựa vào địa chỉ của master để tự động thực hiện quá trình kết nối tới master.

### 5.2.3 Chạy thử nghiệm hệ thống

**5.2.3.1 Thực hiện các chức năng của hệ thống**

Game được cài đặt trên các thiết bị di động hỗ trợ Java thường và các smartphone chạy hệ điều hành Android. Qua quá trình chạy thử nghiệm việc kết nối giữa các thiết bị client và server đều diễn ra thông suốt. Các chức năng đăng nhập và đăng ký đều đạt yêu cầu và không xảy ra lỗi, thông tin đều được cập nhật một cách kịp thời vào cơ sở dữ liệu.

Khi vào chơi game các chức năng như luyện tập, xem top người chơi, chọn bàn, chọn phòng, đọc và gửi tin nhắn, … đều đảm bảo so với yêu cầu đề ra trên cả hai phiên bản. Khi tiến hành chơi trong bàn chơi với 8 người chơi, đã thực hiện kiểm tra với trường hợp 4 điện thoại hỗ trợ Java thường và 4 smartphone Android, thì ván chơi được tiến hành liên tục, đôi khi tín hiệu kết nối bị chậm nhưng ván chơi không bị gián đoạn, 8 người chơi có thể chơi liên tục 3-4 ván chơi (mỗi ván có 10 câu hỏi).

Giữa các thiết bị hỗ trợ Java thường và các smartphone Android không xảy ra tình trạng xung đột giữa hai phiên bản khi cùng kết nối tới một server chung. Với phiên bản Android do phần cứng và tốc độ kết nối mạng khá nhanh nên các tín hiệu truyền về có sự ổn định hơi với các điện thoại thông thường nhưng sự khác biệt không phải là quá lớn (chỉ chênh lệch từ 1s – 3s). Tốc độ trả lời câu hỏi được đảm bảo công bằng giữa các phiên bản vì tốc độ được client gửi lên server và được tính bằng : Thời gian người chơi chọn trả lời ở client trừ đi thời gian người chơi nhận được câu hỏi từ server gửi về.

**5.2.3.2 Kiểm tra khả năng chịu lỗi của hệ thống**

Để đảm bảo khả năng chịu lỗi của hệ thống em đã triển khai cho hệ thống thêm một server dự phòng. Trên server dự phòng này có cài một master server và một slave server. Khi các máy chủ chính hoạt động gặp sự cố, thì sẽ tiến hành khởi chạy server dự phòng, địa chỉ ip của master sẽ được chuyển về server dự phòng, khi đó hệ thống vẫn có thể hoạt động với server dự phòng trong khi đợi khắc phục sự cố ở các server chính.

Trong client cũng có lưu trữ địa chỉ ip và cổng của server dự phòng, khi kết nối tới server chính gặp sự cố và không thể kết nối được thì client sẽ tự kết nối lại tới server dự phòng để vào game. Trong trường hợp ở server chính master gặp sự cố nhưng slave vẫn hoạt động bình thường thì theo thiết kế của khối phục hồi đã đề cập ở chương 3, client sẽ gọi tới một kết nối HttpConnection để lấy được thông tin và địa chỉ của slave để kết nối. Trong quá trình chơi game do mỗi client có một bộ quản lý kết nối riêng trên server nên trong trường hợp client gặp sự cố, bộ kết nối của client tương ứng sẽ bị ngắt khỏi server và các client khác trong game vẫn có thể tiến hành ván chơi bình thường mà không bị gián đoạn.

# KẾT LUẬN

Trong suốt quá trình thực hiện đồ án tốt nghiệp, em đã tập trung và nỗ lực hết sức để xây dựng thành công một hệ thống game online trên di động. Kiến trúc mà em đưa ra cho hệ thống game online là một kiến trúc mang tính khái quát cao, mô hình này có thể áp dụng để xây dựng các game online trên rất nhiều các thiết bị di động khác nhau với nhiều thể loại game khác nhau. Hai phiên bản game online trên J2ME và Android đã được em xây dựng thành công theo mô hình như trên, đặc biệt phiên bản trên J2ME mặc dù bị sự hạn chế về tốc độ kết nối của các thiết bị J2ME (chủ yếu sử dụng kết nối GPRS) vẫn hoạt động khá tốt. Các kĩ thuật chịu lỗi được áp dụng phù hợp và cho hiệu quả tốt, hiện tượng không thể kết nối với hệ thống rất ít khi xảy ra, khi một thành phần kết nối với server bị lỗi, các thành phần khác vẫn có thể hoạt động bình thường. Giao diện, hình ảnh và cách điều khiển của game đơn giản và hợp lý, số lượng câu hỏi lớn và phong phú giúp người chơi luôn cảm thấy hứng thú và tích lũy thêm được nhiều kiến thức.

Bên cạnh những điều đã làm được, đồ án của em vẫn còn những khuyết điểm cần phải được khắc phục. Quá trình kiểm thử chưa được tiến hành đúng đắn nên các lỗi tiềm ẩn trong sản phẩm vẫn còn. Các kĩ thuật chịu lỗi chưa được áp dụng trên toàn hệ thống mà mới chỉ áp dụng trong một mô đun kết nối vì vậy khả năng chịu lỗi của hệ thống là chưa cao. Cơ chế bảo mật và chịu tải của hệ thống cũng chưa được tập trung xây dựng vì vậy hệ thống rất dễ bị tấn công hoặc quá tải.

Trong thời gian tới em sẽ tiếp tục cải tiến và khắc phục những nhược điểm của chương trình. Các lỗi sẽ được tiến hành kiểm thử cẩn thận để phát hiện và sửa lỗi nhằm nâng cao chất lượng chương trình. Các kĩ thuật chịu lỗi sẽ được em tiếp tục nghiên cứu và ứng dụng rộng rãi hơn vào hệ thống. Bên cạnh đó em cũng sẽ nghiên cứu và xây dựng server có khả năng chịu tải cao hơn, các cơ chế bảo mật cũng sẽ được em áp dụng để nâng cao hiệu năng và chất lượng của hệ thống.

# TÀI LIỆU THAM KHẢO

1. Đoàn Văn Ban, *Lập trình Java nâng cao*, NXB Khoa học kĩ thuật, 2006.

2. Ted Neward, *Server-Based Java Programming*, Manning Publications, 2000.

3. Paul Hyde, *Java Thread Programming*, Sams, 1999.

4. Mahmoud Parsian, *JDBC Metadata, MySQL, and Oracle Recipes A Problem-Solution Approach*, Apress, 2006.

5. Sing Li and Jonanthan Knudse, *Beginning J2ME from novice to professional, 3rd Edition*, Apress, 2005.

6. Laura L. Pullum, *Software fault tolerance techniques and implementation*, Artech House, 2001.

7. Israel Koren and C. Mani Krishna, *Fault Tolerant Systems, Morgan Kaufmann Publishers*, 2007.

8. Martin Hiller, *Software Fault-Tolerance Techniques from a Real-Time Systems Point of View*, Department of Computer Engineering Chalmers University of Technology, Sweden, 1996.

9. B.Randell, *System structure for Software Fault Tolerance*, IEEE-Software Eng.,vol. SE-1, pp.220-232, June 1975.

# DANH SÁCH HÌNH VẼ

[Hình 1. Độ tin cậy 14](#_Toc326050743)

[Hình 2. Bốn giai đoạn của chịu lỗi 15](#_Toc326050744)

[Hình 3. Phục hồi lỗi về phía sau (Backward Recovery) 16](#_Toc326050745)

[Hình 4. Phục hồi về phía trước (Forward Recovery) 16](#_Toc326050746)

[Hình 5. Dự phòng phần mềm 19](#_Toc326050747)

[Hình 6. Cặp xử lý 23](#_Toc326050748)

[Hình 7. Kiến trúc cơ bản của khối phục hồi 24](#_Toc326050749)

[Hình 8. Hiệu ứng Domino 27](#_Toc326050750)

[Hình 9. Cấu trúc N-Version Programming 28](#_Toc326050751)

[Hình 10. Các trạng thái của phiên bản. 30](#_Toc326050752)

[Hình 11.a : Thành phần tự kiểm tra với bộ kiểm thử chấp nhận 30](#_Toc326050753)

[Hình 11.b : Thành phần tự kiểm tra gồm một cặp biến thể và bộ so sánh 30](#_Toc326050754)

[Hình 12. Cấu trúc một đơn vị đa dạng. 31](#_Toc326050755)

[Hình 13. Kiến trúc cơ bản của CRB. 32](#_Toc326050756)

[Hình 14: Server tạo Socket theo TCP 33](#_Toc326050757)

[Hình 15: Client tạo socket và thiết lập kết nối theo TCP 34](#_Toc326050758)

[Hình 16: Trao đổi thông tin giữa client và server qua socket theo TCP 35](#_Toc326050759)

[Hình 17: Kết thúc phiên kết nối socket dùng TCP. 36](#_Toc326050760)

[Hình 18: Toàn bộ quá trình kết nối client – server dùng socket theo TCP. 36](#_Toc326050761)

[Hình 19: Server tạo socket theo UDP. 37](#_Toc326050762)

[Hình 20: Client tạo socket theo UDP. 37](#_Toc326050763)

[Hình 21: Trao đổi thông tin giữa client và server theo UDP. 37](#_Toc326050764)

[Hình 22. Kiến trúc server 39](#_Toc326050765)

[Hình 23. Kiến trúc client 41](#_Toc326050766)

[Hình 24. Mô đun kết nối ứng dụng kĩ thuật Recovery Block. 43](#_Toc326050767)

[Hình 25: Biểu đồ use case game “Đấu trường trí tuệ”. 45](#_Toc326050768)

[Hình 26: Các chức năng trong bàn chơi. 47](#_Toc326050769)

[Hình 27: Cơ sở dữ liệu của game. 48](#_Toc326050770)

[Hình 28. Biểu đồ lớp của Master server. 51](#_Toc326050771)

[Hình 29. Biểu đồ lớp của Slave Server. 58](#_Toc326050772)

[Hình 30. Biểu đồ lớp của client. 64](#_Toc326050773)

[Hình 31: Sơ đồ cấu trúc các mô đun của client. 67](#_Toc326050774)

[Hình 32: Các giao diện cơ bản của game trên di động. 68](#_Toc326050775)

[Hình 33: Giao diện giới thiệu game. 69](#_Toc326050776)

[Hình 34: Giao diện đăng ký, đăng nhập. 69](#_Toc326050777)

[Hình 35: Giao diện chọn server và chọn phòng. 70](#_Toc326050778)

[Hình 36: Giao diện chọn bàn chơi. 70](#_Toc326050779)

[Hình 37: Giao diện trong bàn chơi. 71](#_Toc326050780)

[Hình 38a: Giao diện trả lời câu hỏi khi luyện tập. 72](#_Toc326050781)

[Hình 38b: Giao diện trả lời câu hỏi khi thi đấu. 72](#_Toc326050782)

[Hình 39: Giao diện kết quả. 73](#_Toc326050783)

[Hình 40a: Giao diện cửa hàng và Giao diện thông tin cá nhân 73](#_Toc326050784)

[Hình 40b: Giao diện Top người chơi và Hộp thư. 74](#_Toc326050785)

[Hình 40c: Giao diện nạp tiền và xem danh sách người đang chờ. 74](#_Toc326050786)

# 

# DANH SÁCH CÁC BẢNG

[Bảng 1: Mô tả bảng USER 49](#_Toc326051527)

[Bảng 2: Mô tả bảng AVATAR. 49](#_Toc326051528)

[Bảng 3: Mô tả bảng QUESTION. 49](#_Toc326051529)

[Bảng 4: Mô tả bảng MESSAGE. 50](#_Toc326051530)

[Bảng 5: Mô tả bảng FRIEND. 50](#_Toc326051531)

[Bảng 6: Mô tả bảng KIND\_QUESTION. 50](#_Toc326051532)

[Bảng 7. Các phương thức chính của lớp MasterServer. 52](#_Toc326051533)

[Bàng 8. Các phương thức của ConfigReader(MasterServer). 53](#_Toc326051534)

[Bảng 9: Các phương thức của MasterClient. 54](#_Toc326051535)

[Bảng 10: Các phương thức của ConnectionHandler. 55](#_Toc326051536)

[Bảng 11. Các phương thức của Dispatcher trong Master Server. 55](#_Toc326051537)

[Bảng 12: Các phương thức của ReaderThread trong MasterServer. 56](#_Toc326051538)

[Bảng 13: Các phương thức của SlaveServer 59](#_Toc326051539)

[Bảng 14. Các phương thức của ConfigReader(SlaveServer). 59](#_Toc326051540)

[Bảng 15: Các phương thức của SlaveClient. 61](#_Toc326051541)

[Bảng 16: Các phương thức của ConnectionHandler trong SlaveServer. 62](#_Toc326051542)

[Bảng 17. Các phương thức của Dispatcher trong Slave Server. 62](#_Toc326051543)

[Bảng 18: Các phương thức của ReaderThread trong SlaveServer. 63](#_Toc326051544)

[Bảng 19: Các phương thức của GameClient. 65](#_Toc326051545)

[Bảng 20: Các phương thức của ConnectionHandler ở client. 65](#_Toc326051546)

[Bảng 21: Các phương thử của ReaderThread ở client. 65](#_Toc326051547)

[Bảng 22: Các phương thức của WriterThread ở client. 66](#_Toc326051548)