

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

**tìm hiểu emf và gEf, xây dựng công cụ phát sinh mã nguồn cho ứng dụng chạy trên thiết bị wearable**

Giáo viên hướng dẫn: Tôn Long Phước

Người thực hiện MSSV

Nguyễn Như Thuận 14037361

Nguyễn Hữu Thuật 14023091

# MỤC LỤC

[MỤC LỤC 2](#_Toc513973038)

[DANH MỤC CÁC HÌNH VẼ 5](#_Toc513973039)

[DANH MỤC TỪ VIẾT TẮT 8](#_Toc513973040)

[DANH MỤC BẢNG 9](#_Toc513973041)

[LỜI MỞ ĐẦU 10](#_Toc513973042)

[CHƯƠNG 1 : GIỚI THIỆU ĐỀ TÀI 12](#_Toc513973043)

[1.1 Tổng quan 12](#_Toc513973044)

[1.2 Mục tiêu đề tài 13](#_Toc513973045)

[1.2.1 Mục tiêu chung 13](#_Toc513973046)

[1.2.2 Mục tiêu cụ thể 13](#_Toc513973047)

[1.3 Phạm vi đề tài 14](#_Toc513973048)

[CHƯƠNG 2 CƠ SỞ LÝ THUYẾT 15](#_Toc513973049)

[2.1 Eclipse Modeling Framework (EMF) 15](#_Toc513973050)

[2.1.1 EMF (Core) 15](#_Toc513973051)

[2.1.2 Ecore 16](#_Toc513973052)

[2.2 Graphical Editing Framework (GEF) 17](#_Toc513973053)

[2.2.1 Model 18](#_Toc513973054)

[2.2.2 View 19](#_Toc513973055)

[2.2.3 Figures 20](#_Toc513973056)

[2.2.4 Controller 20](#_Toc513973057)

[2.2.5 EditParts 21](#_Toc513973058)

[2.2.6 Request 22](#_Toc513973059)

[2.2.7 Editpolicies 22](#_Toc513973060)

[2.2.8 Command 23](#_Toc513973061)

[2.2.9 Connection 23](#_Toc513973062)

[2.3 Giới thiệu Arduino UNO R3 24](#_Toc513973063)

[2.3.1 Thông số về Arduino UNO R3: 24](#_Toc513973064)

[2.3.2 Vi điều khiển. 24](#_Toc513973065)

[2.3.3 Năng lượng 25](#_Toc513973066)

[2.3.4 Các chân năng lượng. 25](#_Toc513973067)

[2.3.5 Bộ nhớ 25](#_Toc513973068)

[2.3.6 Các chân Arduino UNO R3 26](#_Toc513973069)

[2.3.7 Lưu ý 27](#_Toc513973070)

[2.4 Ứng dụng trên Arduino 28](#_Toc513973071)

[2.5 Máy trạng thái 28](#_Toc513973072)

[2.6 Eclipse plug - in 30](#_Toc513973073)

[CHƯƠNG 3 : PHÂN TÍCH VÀ THIẾT KẾ 31](#_Toc513973074)

[3.1 Đèn LED với Arduino 31](#_Toc513973075)

[3.2 Keypad 4x4 với Arduino 32](#_Toc513973076)

[3.3 Giao diện người dùng 35](#_Toc513973077)

[3.3.1 Giao diện mô tả thiết bị. 35](#_Toc513973078)

[3.3.2 Giao diện mô tả ứng dụng. 35](#_Toc513973079)

[3.4 Sơ đồ lớp Model 36](#_Toc513973080)

[3.5 Thiết kế giao diện người dùng 37](#_Toc513973081)

[3.6 Phân tích cú pháp 38](#_Toc513973082)

[3.7 Bộ phát sinh mã 39](#_Toc513973083)

[3.7.1 Tổng quan kiến trúc 39](#_Toc513973084)

[3.7.2 Giai đoạn 1: Converter - Chuyển đổi tệp nguồn. 40](#_Toc513973085)

[3.7.3 Giai đoạn 2: Device Analyzer - Phân tích các thiết bị 41](#_Toc513973086)

[3.7.4 Giai đoạn 3: State Schema Analyzer – Phần tích lược đồ trạng thái 46](#_Toc513973087)

[3.7.5 Giai đoạn 4: Format – Định dạng mã nguồn 50](#_Toc513973088)

[3.8 Tạo tài liệu hướng dẫn lắp đặt và danh sách thiết bị 50](#_Toc513973089)

[CHƯƠNG 4 : HIỆN THỰC 51](#_Toc513973090)

[4.1 Thiết kế model với EMF 51](#_Toc513973091)

[4.2 Phát sinh model với EMF 54](#_Toc513973092)

[4.2.1 model plug-in 54](#_Toc513973093)

[4.2.2 model.edit plug-in 55](#_Toc513973094)

[4.2.3 model.editor plug-in 56](#_Toc513973095)

[4.2.4 model.test plug-in 56](#_Toc513973096)

[4.3 Xây dựng giao diện người dùng với GEF 57](#_Toc513973097)

[4.3.1 Giao diện cho bo mạch và thiết bị 57](#_Toc513973098)

[4.3.2 Giao diện cho máy trạng thái 57](#_Toc513973099)

[4.4 Xây dựng bộ cú pháp 58](#_Toc513973100)

[4.6 Kiểm chứng 59](#_Toc513973101)

[4.6.1 Bước 1: Tạo dự án 59](#_Toc513973102)

[4.6.2 Bước 2: Thêm và cài đặ thông số cho các thiết bị. 67](#_Toc513973103)

[4.6.3 Bước 2: Mô tả lược đồ trạng thái chương trình. 69](#_Toc513973104)

[4.6.4 Bước 3: Thực hiện chức năng phát sinh mã 70](#_Toc513973105)

[4.6.5 Mã nguồn được phát sinh 71](#_Toc513973106)

[4.6.6 Tài liệu hướng dẫn cho ứng dụng 75](#_Toc513973107)

[4.6.7 Build file mã nguồn vào Arduino UNO R3 76](#_Toc513973108)

[CHƯƠNG 5 : KẾT LUẬN 80](#_Toc513973109)

[5.1.1 Kết quả đạt được 80](#_Toc513973110)

[5.1.2 .Hạn chế của đồ án 80](#_Toc513973111)

[5.1.3 Hướng phát triển 80](#_Toc513973112)

[TÀI LIỆU THAM KHẢO 82](#_Toc513973113)

[PHỤ LỤC 84](#_Toc513973114)

[5.2 Kế hoạch 84](#_Toc513973115)

[5.3 Nhật ký 85](#_Toc513973116)

# DANH MỤC CÁC HÌNH VẼ

[Hình 1: Ecore Model và Java Code [6] 15](#_Toc514488381)

[Hình 2: UML Ecore [7] 16](#_Toc514488382)

[Hình 3: GEF MVC [8] 17](#_Toc514488383)

[Hình 4: GEF Plug-in [9] 18](#_Toc514488384)

[Hình 5: Đối tượng model trong GEF [3] 19](#_Toc514488385)

[Hình 6: Đối tượng view trong GEF [3] 19](#_Toc514488386)

[Hình 7: Cây các đối tượng Figure bởi lightweight system [3] 20](#_Toc514488387)

[Hình 8: Mối liên hệ giữa Mode – EditPart – View [3] 21](#_Toc514488388)

[Hình 9: Command [11] 23](#_Toc514488389)

[Hình 10: Vi điều khiển Arduino UNO R3 [12] 24](#_Toc514488390)

[Hình 11: Các chân Arduino UNO R3 [12] 26](#_Toc514488391)

[Hình 12: Chương trình điều khiển đèn LED nhấp nháy trên Arduino 28](#_Toc514488392)

[Hình 13: Lược đồ máy trạng thái 29](#_Toc514488393)

[Hình 14: Eclipse platform [14] 30](#_Toc514488394)

[Hình 15: Các lắp mạch kết nối LED với Arduino UNO R3 [15] 31](#_Toc514488395)

[Hình 16: Sơ đồ mạch Keypad 4x4 [16] 32](#_Toc514488396)

[Hình 17: Mã khai báo loại keypad sử dụng 33](#_Toc514488397)

[Hình 18: Mã khai báo chân kết nối và đối tượng keypad 33](#_Toc514488398)

[Hình 19: Mã xử lý sự kiện nhấn nút trên keypad 34](#_Toc514488399)

[Hình 20: Sơ đồ lớp model 36](#_Toc514488400)

[Hình 21: Sơ đồ lớp Editor 37](#_Toc514488401)

[Hình 22: Sơ đồ phát sinh mã 39](#_Toc514488402)

[Hình 23: Tệp mainboard.iotw 40](#_Toc514488403)

[Hình 24: Tệp stateSchema.iotw 40](#_Toc514488404)

[Hình 25: Keypad4x4 41](#_Toc514488405)

[Hình 26: Ví dụ về khai báo thư viện cho keypad 41](#_Toc514488406)

[Hình 27: Ví dụ mã định nghĩa cho đèn LED trong Arduino 42](#_Toc514488407)

[Hình 28: Ví dụ mã định nghĩa cho keypad4x4 trong Arduino 42](#_Toc514488408)

[Hình 29: Sơ đồ lớp Device Analyzer 43](#_Toc514488409)

[Hình 30: Sơ đồ trình tự phân tích thiết bị 45](#_Toc514488410)

[Hình 31: Tạo mã với khung trạng thái 46](#_Toc514488411)

[Hình 32: Bộ phân tích cú pháp lệnh 46](#_Toc514488412)

[Hình 33: Sơ đồ ánh xạ các giai đoạn phát sinh mã của lược đồ trạng thái 47](#_Toc514488413)

[Hình 34: Sơ đồ lớp phân tích lược đồ trạng thái 48](#_Toc514488414)

[Hình 35: Sơ đồ trình tự phân tích lược đồ trạng thái 49](#_Toc514488415)

[Hình 36: Thiết kế model sử dung EMF-1 51](#_Toc514488416)

[Hình 37: Thiết kế model sử dung EMF-2 52](#_Toc514488417)

[Hình 38: Thiết kế model sử dung EMF-3 53](#_Toc514488418)

[Hình 39: model plug-in 54](#_Toc514488419)

[Hình 40: model.edit plug-in 55](#_Toc514488420)

[Hình 41: model.editor plug-in 56](#_Toc514488421)

[Hình 42: model.test plug-in 56](#_Toc514488422)

[Hình 43: Giao diện cho bo mạch và thiết bị 57](#_Toc514488423)

[Hình 44: Giao diện cho máy trạng thái 57](#_Toc514488424)

[Hình 45: Tạo project 59](#_Toc514488425)

[Hình 46: Đặt tên project 60](#_Toc514488426)

[Hình 47: Tạo bo mạch cho dự án 61](#_Toc514488427)

[Hình 48: Đặt tên cho bo mạch 62](#_Toc514488428)

[Hình 49: Chọn loại Arduino cho bo mạch 63](#_Toc514488429)

[Hình 50: Mở giao diện cấu hình thiết bị 64](#_Toc514488430)

[Hình 51: Giao diện cấu hình thiết bị 64](#_Toc514488431)

[Hình 52: Chọn loại Object cho lược đồ trạng thái 65](#_Toc514488432)

[Hình 53: Mở giao diện mô tả lược đồ trạng thái 66](#_Toc514488433)

[Hình 54: Giao diện mô tả lược đồ trạng thái. 66](#_Toc514488434)

[Hình 55: Cài đặt thông số cho Keypad 4x4 67](#_Toc514488435)

[Hình 56: Cài đặt thông số cho đèn LED 1 67](#_Toc514488436)

[Hình 57: Cài đặt thông số cho đèn LED2 68](#_Toc514488437)

[Hình 58: Cài đặt thông số cho đèn LED 3 68](#_Toc514488438)

[Hình 59: Lược đồ trạng thái cho ứng dụng điều khiển đèn LED 69](#_Toc514488439)

[Hình 60: Chọn chứ năng phát sinh mã 70](#_Toc514488440)

[Hình 61: Chọn dựa án 70](#_Toc514488441)

[Hình 62: Kết quả sau khi tạo mã 71](#_Toc514488442)

[Hình 63: Mã nguồn được phát sinh phần 1 71](#_Toc514488443)

[Hình 64: Mã nguồn được phát sinh phần 2 72](#_Toc514488444)

[Hình 65: Mã nguồn sau khi phát sinh phần 3 73](#_Toc514488445)

[Hình 66: Mã nguồn sau khi phát sinh phần 4 74](#_Toc514488446)

[Hình 67: Tài liệu cho ứng dụng mô tả 75](#_Toc514488447)

[Hình 68: Mở File mã vừa phát sinh 76](#_Toc514488448)

[Hình 69: Kết nối thiết bị thực tế 77](#_Toc514488449)

[Hình 70: Trạng thái khi nhấn nút số 0 78](#_Toc514488450)

[Hình 71: Trạng thái khi nhấn nút số 1 79](#_Toc514488451)

[Hình 72: Trạng thái ban đầu của 3 đèn LED 79](#_Toc514488452)

[Hình 73: Danh sách các mục trao đổi với GVHD 85](#_Toc514488453)

[Hình 74: Danh mục các công việc 86](#_Toc514488454)

[Hình 75: Cách tổ chức quản lý công việc 86](#_Toc514488455)

# DANH MỤC TỪ VIẾT TẮT

|  |  |
| --- | --- |
| Từ viết tắt | Diễn giải |
| GEF | [Graphical Editing Framework](http://www.eclipse.org/gef) |
| EMF | Eclipse Modeling Framework |
| GUI | Graphical User Interface |
| XMI | XML Metadata Interchange |
| XML | eXtensible Markup Language |
| MVC | Model View Controller |

Bảng 0‑1: Danh mục từ viết tắt

# DANH MỤC BẢNG

[Bảng 0‑1: Danh mục từ viết tắt 8](#_Toc514487191)

[Bảng 2‑1: Thông số Arduino UNO R3 [12] 24](#_Toc514487192)

[Bảng 3‑1: Cách kết nối Keypad với Arduino UNO R3 [16] 32](#_Toc514487193)

[Bảng 3‑2: Mô tả chức năng các lớp Editor 37](#_Toc514487194)

[Bảng 3‑3: Các phương thức tạo mã cho thiết bị 44](#_Toc514487195)

[Bảng 4‑1: Các từ ngữ ánh xạ để mô tà lược đồ trạng thái 58](#_Toc514487196)

[Bảng 4‑2: Thông tin các file được tạo ra khi phát sinh mã 71](#_Toc514487197)

[Bảng 0‑1: Bảng kế hoạch thực hiện đề tài 84](#_Toc514487198)

# LỜI MỞ ĐẦU

1. Tổng quan tình hình nghiên cứu thuộc lĩnh vực của đề tài

Bài toán phát sinh mã nguồn là một trong những bài toán hiện nay có nhiều nhóm nghiên cứu. Các hãng công nghệ cũng quan tâm và phát triển trong lĩnh vực gia công phần mềm. Đây được coi là nhiệm vụ khó khăn vì sự phức tạp của phần mềm, sự phong phú của giao diện người dùng và tính không đồng nhất của các nền tảng phát triển.

Việc phát sinh mã tự động đã có nhiều nhóm nghiên cứu và phát triển, nhưng vẫn còn nhiều hạn chế. Một trong những phần mềm tiên phong trong lĩnh vực phát sinh mã trong lĩnh vực IoT là YAKINDU Statechart Tools. Công cụ này phát sinh theo hướng tiếp cận dựa vào mô hình state machine của ứng dụng [1].

Đề tài này hướng đến việc phát sinh mã cho các ứng dụng chạy trên thiết bị có thể đeo (wearable) dựa vào mô hình hóa và phân tích ngữ nghĩa của mô hình.

2. Ý nghĩa khoa học và thực tiễn của đề tài

Việc phát sinh mã cho ứng dụng chạy trên thiết bị wearable dựa trên sự kết hợp mô hình hóa và xử lý ngữ nghĩa là một hướng tiếp cận mới. Điều này có ý nghĩa thiết thực trong quy trình phát triển công nghệ phần mềm hiện nay.

Sự phát triển của các thiết bị wearable và các ứng dụng, tạo điều kiện cho người dùng tương tác với các hệ thống IoT. Điều này đã thay đổi cách mọi người sử dụng dịch vụ trực tuyến bằng cách giữ cho họ luôn kết nối, dù ở nhà hay đang di chuyển. Thực tế, mỗi thiết bị wearable được thiết kế cho một ứng dụng chuyên dụng như các đối tượng thông minh, thiết bị giám sát lành mạnh, v.v.

Các ứng dụng chạy trên các thiết bị wearable thường đơn giản với các cơ chế xử lý logic. Đồng thời các thiết bị wearable thường là các thiết bị có cấu trúc đơn giản gồm các thành phần như: keypad, bo mạch, cảm biết,….Những công cụ hỗ trợ việc lập trình cho các thiết bị này còn hạn chế và nhiều vấn đề còn bỏ ngỏ.

Với những lý do trên, việc xây dựng công cụ phát sinh mã cho ứng dụng chạy trên thiết bị wearable là cần thiết. Công cụ này giúp các nhà phát triển ứng dụng tự thiết kế, xây dựng ứng dụng một cách nhanh chóng và trực quan. Giúp giảm thời gian, chi phí. Từ đó nhóm tiếp cận đề tài “Tìm hiểu EMF và GEF, xây dựng công cụ phát sinh mã nguồn cho ứng dụng chạy trên thiết bị wearable” cho đồ án tốt nghiệp chuyên ngành Công Nghệ Phần Mềm.

# : GIỚI THIỆU ĐỀ TÀI

## Tổng quan

Đề tài này tiếp cận theo hướng phát sinh mã nguồn bằng cách mô hình hóa và phân tích ngữ nghĩa. Nhằm phát sinh mã nguồn C cho ứng dụng chạy trên thiết bị wearable. Người dùng có thể thiết kế trực quan và phát sinh mã nguồn. Với các xử lý phức tạp, mã nguồn được phát sinh cần được các nhà phát triển tùy chỉnh thêm để hoàn chỉnh mã nguồn. Đề tài gồm các công việc sau:

* Tìm hiểu về Eclipse Modeling Framework (EMF) [2] nhằm xây dựng meta model: Các thiết bị thường được cập nhật tính năng thường xuyên như tích hợp thêm vi xử lý, tính năng… từ nhà sản xuất. Nên cần một công cụ giúp quản lý model có thể tùy biến, dễ dàng cập nhật những thay đổi đó. EMF đáp ứng được những tiêu chí trên. Nó hỗ trợ các API, cơ chế lưu trữ các meta model dưới dạng xml và kết hợp tốt với GEF.
* Tìm hiểu công nghệ Graphical Editing Framework (GEF)[3] nhằm thiết kế giao diện đồ họa người dùng (GUI). Gồm hai trình soạn thảo đồ họa cho phép người dùng thao tác và đặc tả ứng dụng. Để tạo giao diện một cách chuyên nghiệp, nhanh chóng và tùy chỉnh dễ dàng thì GEF là sự lựa chọn thích hợp. Đây là framework hỗ trợ xây dựng trình soạn thảo cho phép nhà phát triển tùy biến, dễ dàng tương thích với các loại model khác nhau nhờ kiến trúc MVC và cung cấp đầy đủ các tính năng của một trình soạn thảo đồ họa. GEF cũng được sử dụng nhiều ví dụ như Ant Visualization [4], Solunar GmbH [5]…
* Kết hợp giữa model và giao diện đồ họa.
* Xây dựng bộ cú pháp cho phép ánh xạ các mô tả trong lược đồ trạng thái với thiết bị trên bo mạch.
* Thiết kế ứng dụng chạy trên thiết bị wearable đơn giản.
* Kiểm chứng mã sau khi được phát sinh.

## Mục tiêu đề tài

### Mục tiêu chung

Xây dựng công cụ WAGEN Tools cho phép thiết kế trực quan và phát sinh mã nguồn cho ứng dụng điều khiển đèn LED bằng Keypad4x4 với bo mạch Arduino UNO R3.

### Mục tiêu cụ thể

1. Tìm hiểu công nghệ Eclipse Modeling Framework - EMF để xây dựng model.
2. Tìm hiểu công nghệ Graphical Editing Framework – GEF để thiết kế giao diện đồ họa người dùng (GUI).
3. Tìm hiểu ứng dụng trên Arduino.
4. Tìm hiểu máy trạng thái.
5. Tìm hiểu Eclipse plug-in.
6. Xây dựng model sử dụng EMF

* Đối tượng bo mạch: Arduino UNO R3.
* Đối tượng thiết bị đầu vào: Keypad4x4.
* Đối tượng thiết bị đầu ra: đèn LED.
* Các đối tượng cho máy trạng thái.

1. Xây dựng giao diện cho người dùng (GUI) sử dụng GEF.

* Tạo trình soạn thảo đồ họa cho phần thiết bị kết nối với bo mạch.
* Tạo trình soạn thảo đồ họa cho phần máy trạng thái.

1. Kết hợp model với giao diện đồ họa.
2. Xây dựng bộ cú pháp cho phép ánh xạ các mô tả trong lược đồ trạng thái với thiết bị trên bo mạch.
3. Xây dựng bộ phát sinh mã nguồn cho ứng dụng chạy trên thiết bị wearable.
4. Thiết kế ứng dụng điều khiển đèn LED bằng Keypad4x4 với bo mạch Arduino UNO R3. Kiểm chứng mã nguồn được phát sinh.

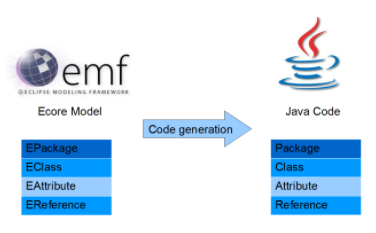
## Phạm vi đề tài

1. Tìm hiểu về EMF để xây dựng model.
2. Tìm hiểu GEF để xây dựng giao diện đồ họa người dùng.
3. Kết hợp EMF và GEF để tạo công cụ WAGEN Tools chạy trên nền tảng Eclipse Luna.
4. Chức năng phát sinh mã C.
5. Xây dựng bộ ánh xạ mô tả từ lược đồ trạng thái với các thiết bị trên bo mạch.
6. Xây dựng ứng dụng điều khiển đèn LED bằng keypad 4x4 với bo mạch Arduino UNO R3.
7. Kiểm chứng mã nguồn được phát.

# CƠ SỞ LÝ THUYẾT

## Eclipse Modeling Framework (EMF)

EMF [2] dùng để tạo mã cho các công cụ và những ứng dụng khác dựa trên mô hình dữ liệu có cấu trúc. Từ mô tả trong XMI, EMF cung cấp công cụ và hỗ trợ runtime để tạo ra một tập các lớp Java cho mô hình, cùng với một bộ các lớp adapter cho phép xem và chỉnh sửa mô hình dựa trên lệnh, và một trình soạn thảo cơ bản.



Hình : Ecore Model và Java Code [6]

### EMF (Core)

EMF core gồm có ba phần cơ bản:

EMF [2]: EMF framework core bao gồm meta model (Ecore) để mô tả cho model và hỗ trợ model gồm: thông báo runtime, hỗ trợ serialization XMI mặc định và rất nhiều API để thao tác với các đối tượng EMF.

EMF.Edit [2]: EMF.Edit là framework cung cấp nhiều lớp dùng chung để xây dựng trình soạn thảo cho các model EMF. Nó cung cấp:

1. Cung cấp các lớp nội dung (content) và nhãn (label), thuộc tính và những lớp tiện ích cho phép model EMF có thể được trình bày trên giao diện đồ họa và các bảng thuộc tính.
2. Một tập các lệnh, bao gồm một tập hợp các lớp thực thi để xây dựng trình soạn thảo và hỗ trợ tự động thao tác undo, redo.

EMF.Codegen [2]: tập hợp mọi thứ cần thiết để xây dựng một trình soạn thảo hoàn chỉnh cho model EMF. Bao gồm một GUI từ việc tạo ra phiên bản mà các thuộc tính có thể được tùy chọn.

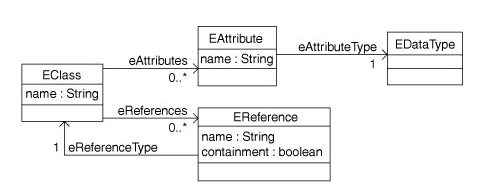
Mã khi được tạo ra gồm:

1. Model: cung cấp các Java interfaces và các lớp implementation cho tất cả các lớp trong mô hình, cùng với factory và một package với các lớp implementation.
2. Adapter: Tạo các lớp implementation (gọi là ItemProvider) để phù hợp với các lớp mô hình để sửa và hiện thị.
3. Editor: Tạo Editor phù hợp với cấu trúc ban đầu và đóng vai trò là điểm xuất phát để bắt đầu tùy biến.

Tất cả trình phát sinh mã hỗ trợ có thể phát sinh lại mã trong khi vẫn duy trì các sửa đổi của người dùng. Trình phát sinh mã có thể gọi thông qua GUI hoặc lệnh.

### Ecore

EMF tạo ra mã java dựa vào việc định nghĩa mô hình theo Ecore, về cơ bản là tập hợp các UML.



Hình : UML Ecore [7]

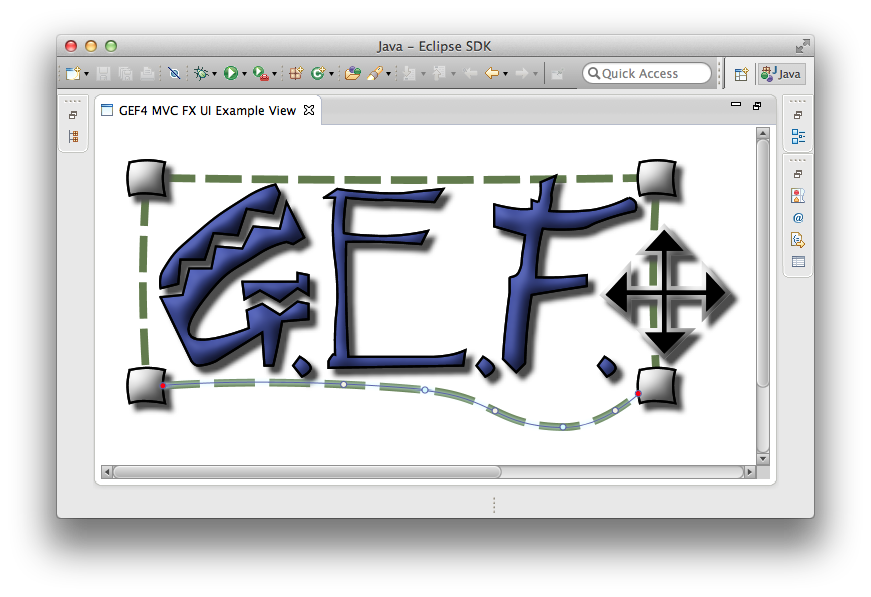
Như hình ta thấy cần bốn lớp Ecore của mô hình:

* EClass [7] : được sử dụng để đại diện cho mô hình class. Nó có tên, có nhiều hoặc không có các thuộc tính và có nhiều hoặc không có các tham chiếu.
* EAttribute [7] : được sử dụng để đại diện cho một thuộc tính được mô hình hóa. Nó có tên và loại.
* EReference [7] : được đại diện cho sự kết hợp giữa các class. Nó có một tên, containment, một loại tham chiếu (target) tới một lớp khác.
* EDataType [7] : được sử dụng để đại diện cho loại của một thuộc tính. Một kiểu dữ liệu có thể là một kiểu nguyên thủy như int hoặc float hoặc một kiểu đối tượng như java.util.Date.

## Graphical Editing Framework (GEF)

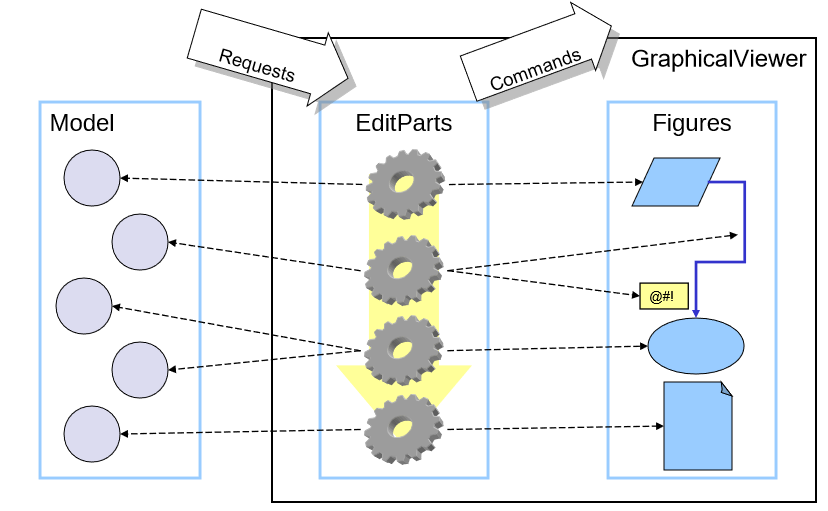
Eclipse Graphical Editing Framework (GEF) cung cấp các công cụ cho người dùng cuối tích hợp Eclipse dưới dạng một trình soạn thảo Graphviz (trình soạn thảo DOT, chế độ xem DOT Graph) [1] cũng như các thành phần khuôn khổ (Common, Geometry, FX, MVC, đồ thị, Giao diện, Zest, DOT và Cloudio) để tạo ra các ứng dụng khách Java dạng đồ họa, tích hợp Eclipse hoặc độc lập.

Có thể hiểu đơn giản là GEF cung cấp các tiện ích cho phép xây dựng một trình soạn thảo với các thực thể đồ họa và các tương tác với thực thể đồ họa đây như kéo, thả, chỉnh sửa kích thước... mà không cần mất công sức xây dựng lại từ đầu, nó có thể được tích hợp cho Eclipse hoặc độc lập.



Hình : GEF MVC [8]

GEF vận hành theo mô hình MVC (Model – View - Controller).

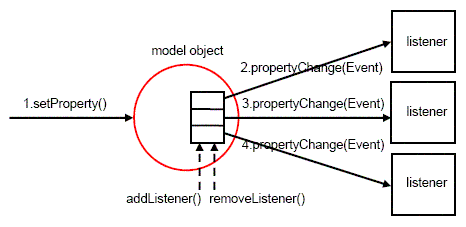


Hình : GEF Plug-in [9]

### Model

Model là vùng lưu trữ dữ liệu có cấu trúc được sửa đổi trong quá trình tương tác với các đối tượng đồ họa tương ứng tại View:

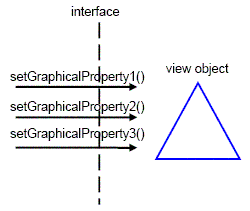
* Model lưu trữ tất cả những gì mà có thể tùy chỉnh hoặc được biểu diễn lên cho người dùng: Cụ thể đưa ra các dữ liệu trực quan [10].
* Cung cấp cách để lưu trữ model: Đảm bảo rằng khi tắt giao diện các thông tin vẫn được lưu trữ và khi mở lại giao diện thì các thông tin được hiển thị lại lên giao diện [10].
* Model phải không liên quan gì đến view hay controller: Không có bất kỳ sự tham chiếu nào của model tới view hay controller. Trong một số trường hợp bỏ một vài view hoặc controller, nếu model tham chiếu đến view hoặc controller ta sẽ mất các figure hoặc EditPart cho model đó [10].
* Cung cấp phương thức để lắng nghe sự thay đổi của model: Điều này cho phép controller phản ứng với các thay đổi và điều chỉnh hiển thị thích hợp lên view. Do không có sự tham chiếu của model của controller nên chỉ có cách là đăng ký cho controller với model như là người nhận sự kiện bằng cách đăng ký thông báo sự kiện thay đổi được định nghĩa trong package java.bean [10].



Hình : Đối tượng model trong GEF [3]

### View

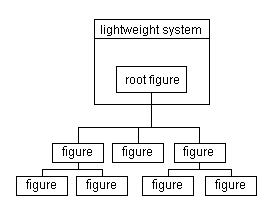
View là tập hợp các đối tượng để tạo ra giao diện đồ họa. Cũng có thể hiểu View là một thể hiện của Model dưới dạng đồ họa. GEF cung cấp hỗ trợ Draw2d Figure [10] cho việc vẽ các đối tượng đồ họa.



Hình : Đối tượng view trong GEF [3]

### Figures

Là đối tượng của Draw2d. Một giao diện đồ họa Draw2d được định nghĩa bởi cây các đối tượng Figure sử dụng bởi lightweight system [10].



Hình : Cây các đối tượng Figure bởi lightweight system [3]

Từ các đối tượng Figure đơn giản, có thể vẽ được các hình phức tạp bằng cách kết hợp nhiều đối tượng Figure.

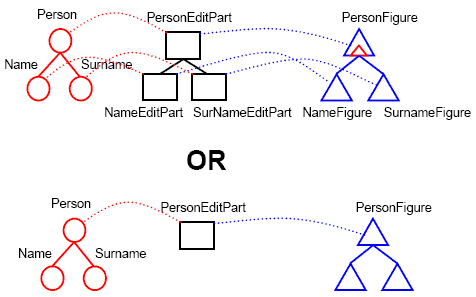
Layout Management: GEF cung cấp framework quản lý layout. Công việc của nó là xử lý các layout của figures con của Draw2d IFigure. Công việc của người dùng là quyết định dùng trình quản lý layout nào cho mỗi figure và figure con.

### Controller

Vì model không giữ bất kỳ tham chiếu nào cho view ngược lại. Do đó controller đảm nhiệm việc liên kết giữa model và view. Trong GEF, các controller là các lớp con của EditPart. [10]

### EditParts

EditPart liên kết từng đối tượng model với view nó.



Hình : Mối liên hệ giữa Mode – EditPart – View [3]

EditPart lắng nghe giữa Model và View, thu thập dữ liệu, nhận các request và thực thi những điều khiển. Với mỗi phần độc lập của model cần được xác định bởi controller. Các yếu tố phụ thuộc vào thao tác của người dùng bất cứ thứ gì được chọn hoặc xóa đều cần có phần EditPart cho riêng nó. Vai trò của EditPart là hiểu model lắng nghe sự kiện về sự thay đổi, cập nhật view tương ứng với thay đổi.

Mọi phần của model có thể chỉnh sửa riêng biệt cần được liên kết tới một EditPart. Điều này có nghĩa là thường có sự mapping one-for-one giữ các lớp trong model với các lớp của EditPart. Trong hầu hết các trường hợp một EditPart cũng là một GraphicalEditPart có nghĩa là quản lý cho một model component và là quản lý cho một view component. Bởi vì model và view hoàn toàn tách rời nên tất cả sự phối hợp giữa view và model phải được quản lý bởi EditPart. [10]

Sự liên kết này có thể chia làm 2 phần:

* Đảm nhiệm vai trò lắng nghe những thay đổi của model để có thể chuyển đến view bằng các phương thức gọi các layout liên quan.
* Cung cấp phương tiện mà các tương tác người dùng có thể diễn giải và thông báo cho những thay đổi của model.
* Quản lý các phương thức chỉnh sửa trực tiếp. Nơi người dùng gõ văn bản trực tiếp vào các nội dung của control.

### Request

Request biến đổi các thao tác của người dùng thành các request. Giúp ứng dụng có thể làm theo kiểu hướng đối tượng. Khi người dùng thao tác với ứng dụng GEF's behind-the-scenes sẽ tạo ra các request object.

Các kiểu tương tác khác nhau tạo ra các loại request khác nhau. Các loại request này được định nghĩa trong GEF API và tài liệu hướng dẫn. Những request này đóng gói thông tin mà ứng dụng cần để biến đổi tương tác của người dùng thành những thay đổi đối với model [10].

### Editpolicies

EditPart không xử lý trực tiếp các chỉnh sửa. Thay vào đó sử dụng EditPolicies. Mỗi EditPolicy tập trung chỉnh sửa nhiệm vụ đơn hoặc nhóm các nhiệm vụ có liên quan. Điều này cho phép các xử lý có thể được sử dụng lại trong nhiều EditPart khác nhau. [9]

Khi có bất kỳ sự chỉnh sửa trên view được gọi (ngoại trừ lệnh performRequest()), phần EditPart ủy quyền cho các policy của nó để đáp ứng yêu cầu. Tùy thuộc vào phương pháp, EditPart có thể dừng ở policy đầu tiên xử lý request hoặc cho phép mỗi policy tham gia vào [9].

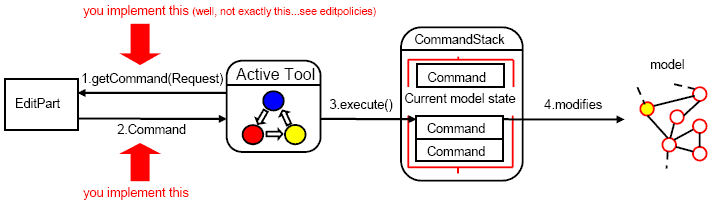
Trong quá trình tạo EditPart, createEditPolicies() được gọi và EditPart nên được gọi tới các policy thích hợp. EditPolicies được cài đặt bằng các roles, là các định danh mà được sử dụng như keys. GEF cung cấp một số định danh cho các roles được dùng phổ biến. Roles đóng vai trò quan trọng khi editpart cần dùng các policies đề thực hiện thao tác xóa, di chuyển. GEF cũng cung cấp rất nhiều policies để sử dụng các roles. Nhiều policies này phải được mở rộng để phù hợp cho model của ứng dụng [9].

### Command

Chịu trách nhiệm thực hiện các thay đổi mô hình, do đó cung cấp các chức năng hoàn tác [10].

Các command là một phần của model. Chúng không phải là model, nhưng nó là phương tiện chỉnh sửa các model. Các Command được dùng để thực hiện tất cả các thay đổi, hoàn tác của người dùng. Các Command chỉ giao tiếp với model [10].

Thay vì sửa model trực tiếp. GEF yêu cầu phải làm điều đó thông qua các Command. Mỗi Command nên thực thi lệnh và hoàn tác các thay đổi với model hoặc 1 phần model. Bằng cách này GEF tự động hỗ trợ undo/redo cho model [10].



Hình : Command [11]

Nó cung cấp các phương thức chính sau:

* execute(): Thực thi các thay đổi với model.
* undo(): Đảo ngược lại các thực thi của execute().
* redo(): Thực thi lại một lệnh đã được thực thi.
* canExecute(): Xác định các điều kiện để một lệnh chó thể được thực thi.
* canUndo(): Xác định xem có thể undo hay không.
* canRedo(): Xác định xem có thể redo hay không.

### Connection

Connection được biểu diễn bằng một liên kết (một đường kẻ) giữa các biểu diễn trực quan của hai đối tượng mô hình khác được gọi là nguồn và đích. Các đối tượng mô hình có khả năng là một nguồn hoặc một đích được gọi là các nút [11].

## Giới thiệu Arduino UNO R3

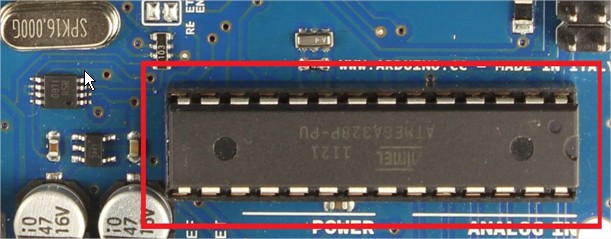
Arduino UNO là một trong những dòng Arduino dùng để lập trình phổ biến nhất. Hiện nay Arduino UNO đã phát triển đến thế hệ thứ 3 (R3). [12]

### Thông số về Arduino UNO R3:

|  |  |
| --- | --- |
| Vi điều khiển | ATmega328 họ 8bit |
| Điện áp hoạt động | 5V DC (chỉ được cấp qua cổng USB) |
| Tần số hoạt động | 16 MHz |
| Dòng tiêu thụ | khoảng 30mA |
| Điện áp vào khuyên dùng | 7-12V DC |
| Điện áp vào giới hạn | 6-20V DC |
| Số chân Digital I/O | 14 (6 chân hardware PWM) |
| Số chân Analog | 6 (độ phân giải 10bit) |
| Dòng tối đa trên mỗi chân I/O | 30 mA |
| Dòng ra tối đa (5V) | 500 mA |
| Dòng ra tối đa (3.3V) | 50 mA |
| Bộ nhớ flash | 32 KB (ATmega328) với 0.5KB dùng bởi bootloader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |

Bảng 2‑1: Thông số Arduino UNO R3 [12]

### Vi điều khiển.



Hình : Vi điều khiển Arduino UNO R3 [12]

Arduino UNO [12] có thể sử dụng 3 vi điều khiển họ 8bit AVR là ATmega8, ATmega168, ATmega328. Bộ vi điều khiển này có thể xử lí những tác vụ đơn giản như điều khiển đèn LED nhấp nháy, xử lí tín hiệu cho xe điều khiển từ xa, làm một trạm đo nhiệt độ - độ ẩm và hiển thị lên màn hình LCD…

Thiết kế tiêu chuẩn của Arduino UNO [12] sử dụng vi điều khiển ATmega328. Tuy nhiên nếu yêu cầu phần cứng không cao, các loại vi điều khiển khác có chức năng tương đương nhưng rẻ hơn như ATmega8 (bộ nhớ flash 8KB) hoặc ATmega168 (bộ nhớ flash 16KB).

### Năng lượng

Arduino UNO [12] có thể được cấp nguồn 5V thông qua cổng USB hoặc cấp nguồn ngoài với điện áp khuyên dùng là 7-12V DC và giới hạn là 6-20V. Thường thì cấp nguồn bằng pin vuông 9V là hợp lí. Nếu cấp nguồn vượt quá ngưỡng giới hạn trên, Arduino UNO sẽ bị hỏng.

### Các chân năng lượng.

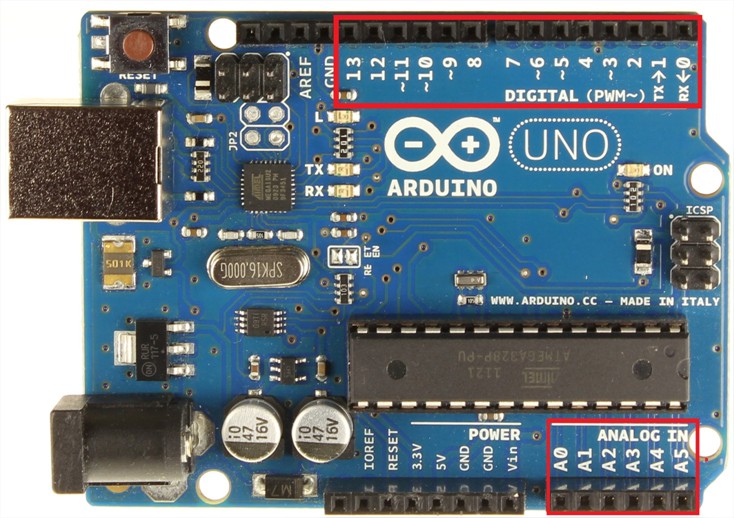
* GND (Ground): cực âm của nguồn điện cấp cho Arduino UNO. [12]
* 5V: cấp điện áp 5V đầu ra. Dòng tối đa cho phép ở chân này là 500mA.
* 3.3V: cấp điện áp 3.3V đầu ra. Dòng tối đa cho phép ở chân này là 50mA.
* Vin (Voltage Input): để cấp nguồn ngoài cho Arduino UNO, bạn nối cực dương của nguồn với chân này và cực âm của nguồn với chân GND.
* IOREF: điện áp hoạt động của vi điều khiển trên Arduino UNO có thể được đo ở chân này. Và dĩ nhiên nó luôn là 5V. Mặc dù vậy không được lấy nguồn 5V từ chân này để sử dụng bởi chức năng của nó không phải là cấp nguồn.
* RESET: việc nhấn nút Reset trên board để reset vi điều khiển tương đương với việc chân RESET được nối với GND qua 1 điện trở 10KΩ.

### Bộ nhớ

Vi điều khiển Atmega328 tiêu chuẩn cung cấp cho người dùng:

* 32KB bộ nhớ Flash: những đoạn lệnh sẽ được lưu trữ trong bộ nhớ Flash của vi điều khiển. Thường thì sẽ có khoảng vài KB trong số này sẽ được dùng cho bootloader.
* 2KB cho SRAM (Static Random Access Memory): giá trị các biến khai báo khi lập trình sẽ lưu ở đây. Việc khai báo càng nhiều biến thì càng cần nhiều bộ nhớ RAM. Khi mất điện, dữ liệu trên SRAM sẽ bị mất.
* 1KB cho EEPROM (Electrically Erasable Programmable Read Only Memory): đây giống như một chiếc ổ cứng mini – nơi đọc và ghi dữ liệu mà không phải lo bị mất khi cúp điện giống như dữ liệu trên SRAM.

### Các chân Arduino UNO R3



Hình : Các chân Arduino UNO R3 [12]

Arduino UNO [12] có 14 chân digital dùng để đọc hoặc xuất tín hiệu. Chúng chỉ có 2 mức điện áp là 0V và 5V với dòng vào/ra tối đa trên mỗi chân là 40mA. Ở mỗi chân đều có các điện trở pull-up từ được cài đặt ngay trong vi điều khiển ATmega328 (mặc định thì các điện trở này không được kết nối).

Một số chân digital có các chức năng đặc biệt như sau:

* 2 chân Serial: 0 (RX) và 1 (TX): dùng để gửi (transmit – TX) và nhận (receive – RX) dữ liệu TTL Serial. Arduino Uno có thể giao tiếp với thiết bị khác thông qua 2 chân này. Kết nối bluetooth thường thấy nói nôm na chính là kết nối Serial không dây. Nếu không cần giao tiếp Serial, thì không nên sử dụng 2 chân này nếu không cần thiết
* Chân PWM (~): 3, 5, 6, 9, 10, và 11: cho phép xuất ra xung PWM với độ phân giải 8bit (giá trị từ 0 → 28-1 tương ứng với 0V → 5V) bằng hàm analogWrite(). Nói một cách đơn giản, các chân này có thể được điều chỉnh điện áp ra ở chân này từ mức 0V đến 5V thay vì chỉ cố định ở mức 0V và 5V như những chân khác.
* Chân giao tiếp SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ngoài các chức năng thông thường, 4 chân này còn dùng để truyền phát dữ liệu bằng giao thức SPI với các thiết bị khác.
* LED 13: trên Arduino UNO có 1 đèn led màu cam (kí hiệu chữ L). Khi bấm nút Reset, đèn này nhấp nháy để báo hiệu. Nó được nối với chân số 13. Khi chân này được người dùng sử dụng, LED sẽ sáng.
* Arduino UNO có 6 chân analog (A0 → A5) cung cấp độ phân giải tín hiệu 10bit (0 → 210-1) để đọc giá trị điện áp trong khoảng 0V → 5V. Với chân AREF trên board, có thể đưa vào điện áp tham chiếu khi sử dụng các chân analog. Tức là nếu cấp điện áp 2.5V vào chân này thì có thể dùng các chân analog để đo điện áp trong khoảng từ 0V → 2.5V với độ phân giải vẫn là 10bit.
* Đặc biệt, Arduino UNO có 2 chân A4 (SDA) và A5 (SCL) hỗ trợ giao tiếp I2C/TWI với các thiết bị khác.

### Lưu ý

Arduino UNO không có bảo vệ cắm ngược nguồn vào. Do đó phải hết sức cẩn thận, kiểm tra các cực âm – dương của nguồn trước khi cấp cho Arduino UNO. Việc làm chập mạch nguồn vào của Arduino UNO [12] sẽ biến nó thành một miếng nhựa chặn giấy. Vì vậy nên dùng nguồn từ cổng USB nếu có thể.

Các chân 3.3V và 5V trên Arduino là các chân dùng để cấp nguồn ra cho các thiết bị khác, không phải là các chân cấp nguồn vào. Việc cấp nguồn sai vị trí có thể làm hỏng board. Điều này không được nhà sản xuất khuyến khích.

Cấp nguồn ngoài không qua cổng USB cho Arduino UNO [12] với điện áp dưới 6V có thể làm hỏng board.

Cấp điện áp trên 13V vào chân RESET trên board có thể làm hỏng vi điều khiển ATmega328.

Cường độ dòng điện vào/ra ở tất cả các chân Digital và Analog của Arduino UNO [12] nếu vượt quá 200mA sẽ làm hỏng vi điều khiển.

Cấp điệp áp trên 5.5V vào các chân Digital hoặc Analog của Arduino UNO [12] sẽ làm hỏng vi điều khiển.

Cường độ dòng điện qua một chân Digital hoặc Analog bất kì của Arduino UNO [12] vượt quá 40mA sẽ làm hỏng vi điều khiển. Do đó nếu không dùng để truyền nhận dữ liệu, bạn phải mắc một điện trở hạn dòng.

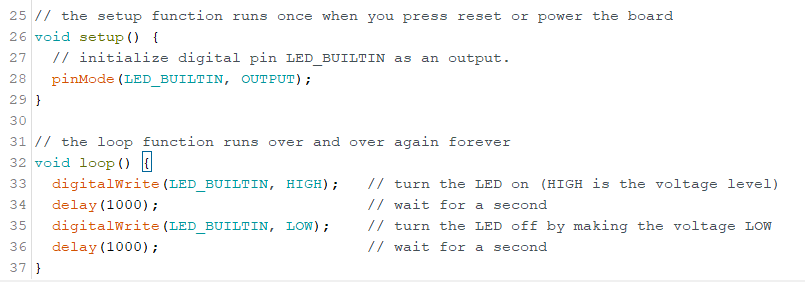
## Ứng dụng trên Arduino

Ở đây đề cập đến mã chương trình của Arduino, một chương trình Arduino được viết bằng ngôn ngữ lập trình C/C++. Nó có một IDE là Arduino IDE đi kèm với thư viện phần mềm giúp cho các thao tác input/output được dễ dàng.

Một chương trình arduino có hai phần chính:

* setup(): hàm này chạy mỗi khi khởi động một chương trình, dùng cho thiết lập các cài đặt.
* loop():  hàm này được gọi lặp lại cho đến khi bo mạch tắt nguồn.

Dưới đây là một ví dụ đơn giản cho một chương trình arduino điểu khiển một đèn LED nhấp nháy.

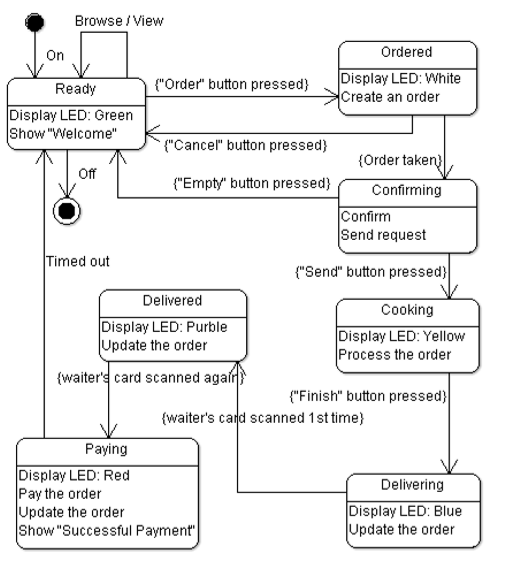


Hình : Chương trình điều khiển đèn LED nhấp nháy trên Arduino

## Máy trạng thái

Máy trạng thái hữu hạn (finite-state machine FSM) hoặc Máy tự động trạng thái hữu hạn (finite-state automaton FSA), hoặc là máy tự động hữu hạn, hoặc gọi đơn giản là máy trạng thái, là một mô hình tính toán. Nó là một máy trừu tượng luôn có trạng thái nằm trong tổng hữu hạn các trạng thái tại bất kỳ thời điểm nào. Máy trạng thái hữu hạn có thể chuyển từ trạng thái này sang trạng thái khác để phù hợp với đầu vào; sự thay đổi này được gọi là quá trình chuyển đổi. Máy trạng thái hữu hạn được xác định bởi danh sách các trạng thái của nó, trạng thái khởi đầu, và các điều kiện cho từng sự chuyển đổi trạng thái [13].

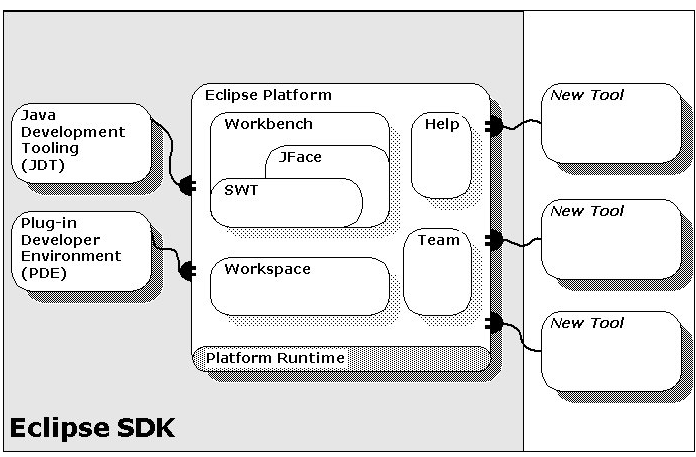
Hành vi của máy trạng thái có thể được quan sát qua nhiều thiết bị hiện đại, đó là việc thực hiện một chuỗi các hành động định trước tùy vào chuỗi sự kiện mà chúng được lập trình. [13]



Hình : Lược đồ máy trạng thái

## Eclipse plug - in

Theo định nghĩa của Eclipse documentation - Previous Release thì plug-in là bó cấu trúc của mã hoặc dữ liệu đóng góp cho hệ thống. Chức năng có thể được đóng góp dưới dạng thư viện mã (các lớp Java với API công cộng), platform extensions, hoặc thậm chí là tài liệu. Plug-ins có thể định nghĩa các extension point, các địa điểm được xác định rõ, nơi các trình cắm thêm khác có thể bổ sung chức năng. [14]

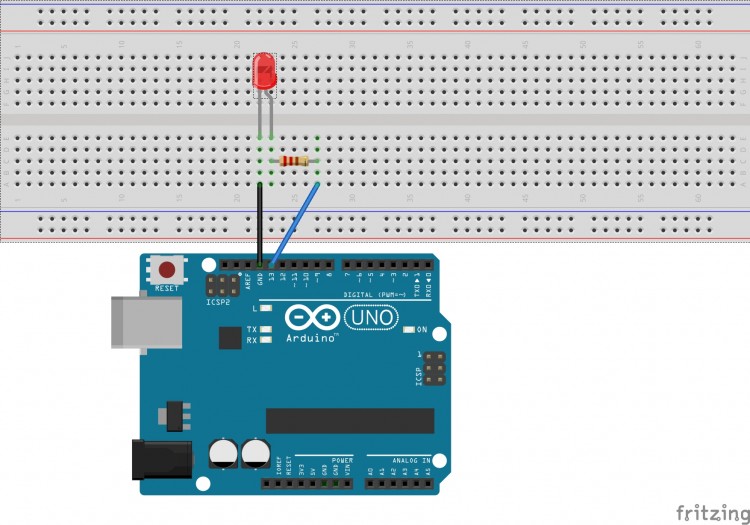


Hình : Eclipse platform [14]

# : PHÂN TÍCH VÀ THIẾT KẾ

## Đèn LED với Arduino

Arduino UNO R3 có thể kết nối với nhiều đèn LED khác nhau.



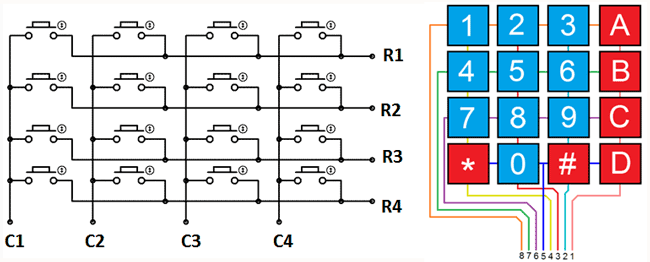
Hình : Các lắp mạch kết nối LED với Arduino UNO R3 [15]

Viết chương trình điểu khiển. Đầu tiên phải khai báo chân mà Arduino sử dụng để điều khiển đèn LED. Trong ví dụ trên: LED kết nối với Arduino UNO R3 với chân 13. Các lệnh khai báo được đặt trong hàm setup() của chương trình.

Lệnh xác định chần output: pinMode(13, OUTPUT);

Lệnh bật đèn LED: digitalWrite(13,HIGH); lệnh này sẽ cấp một điện thế là 5V vào chân số Digital 13. Điện thế sẽ đi qua điện trở 220ohm rồi đến đèn LED (sẽ làm nó sáng). Dể tắt đèn LED sử dụng lệnh: digitalWrite(13,LOW);

## Keypad 4x4 với Arduino



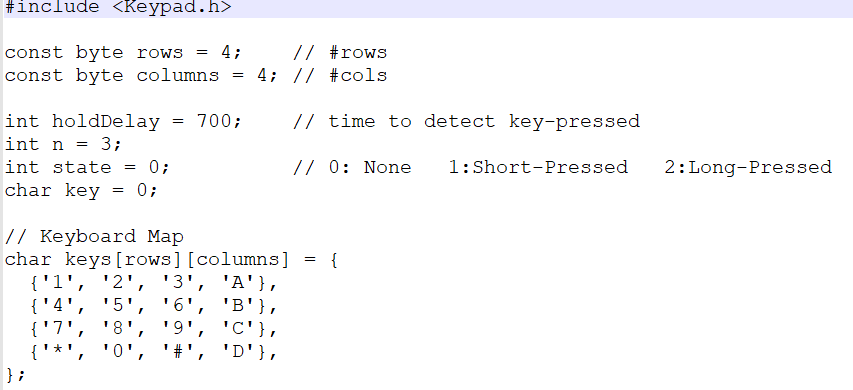
Hình : Sơ đồ mạch Keypad 4x4 [16]

Arduino UNO R3 chỉ có thể kết nối với 1 keypad. Để keypad có thể hoạt động cần sử dụng thư viện Keypad.h. Với Keypad 4x4 cần dùng 8 chân để kết nối với Arduino. Dưới đây là cách nối dây:

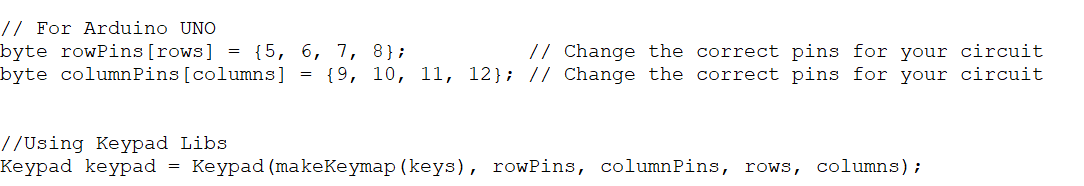
|  |  |
| --- | --- |
| **Arduino** | **Keypad** |
| 5 | R1 |
| 6 | R2 |
| 7 | R3 |
| 8 | R4 |
| 9 | C1 |
| 10 | C2 |
| 11 | C3 |
| 12 | C4 |

Bảng 3‑1: Cách kết nối Keypad với Arduino UNO R3 [16]

Tiếp theo là mã giao tiếp với keypad.

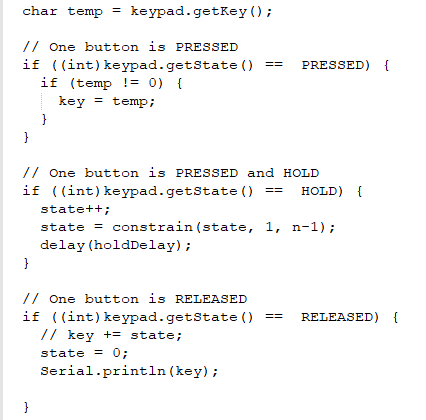


Hình : Mã khai báo loại keypad sử dụng



Hình : Mã khai báo chân kết nối và đối tượng keypad

Xử lý sự kiện nhấn nút.



Hình : Mã xử lý sự kiện nhấn nút trên keypad

## Giao diện người dùng

Muốn kết nối và điều khiển thiết bị với Arduino UNO R3. Cần thực hiện 2 bước sau:

1. Kết nối, khai báo sử dụng thiết bị với Arduino UNO R3.
2. Sử dụng các hàm điều khiển thiết.

Vì vậy cần 2 giao diện để thực hiện 2 bước trên.

### Giao diện mô tả thiết bị.

Trong giao diện này người dùng có thể thêm các thiết bị mong muốn cho chương trình của họ vào. Giao diện hỗ trợ các chức năng:

1. Cho phép người dùng tạo, xóa thiết bị bằng giao diện đồ họa.
2. Kiểm tra, thực hiện việc khai báo kết nối thiết bị đó với Arduino.
3. Gợi ý kết nối các chân sao cho phù hợp với thiết bị.
4. Thông báo khi không thể kết nối thiết bị với Arduino.
5. Cho phép người dùng có thể thiết đặt lại kết nối, đặt Id cho thiết bị phục vụ cho việc gọi lại thiết bị đó khi mô tả ứng dụng.
6. Hỗ trợ các chức năng cơ bản của giao diện đồ họa.

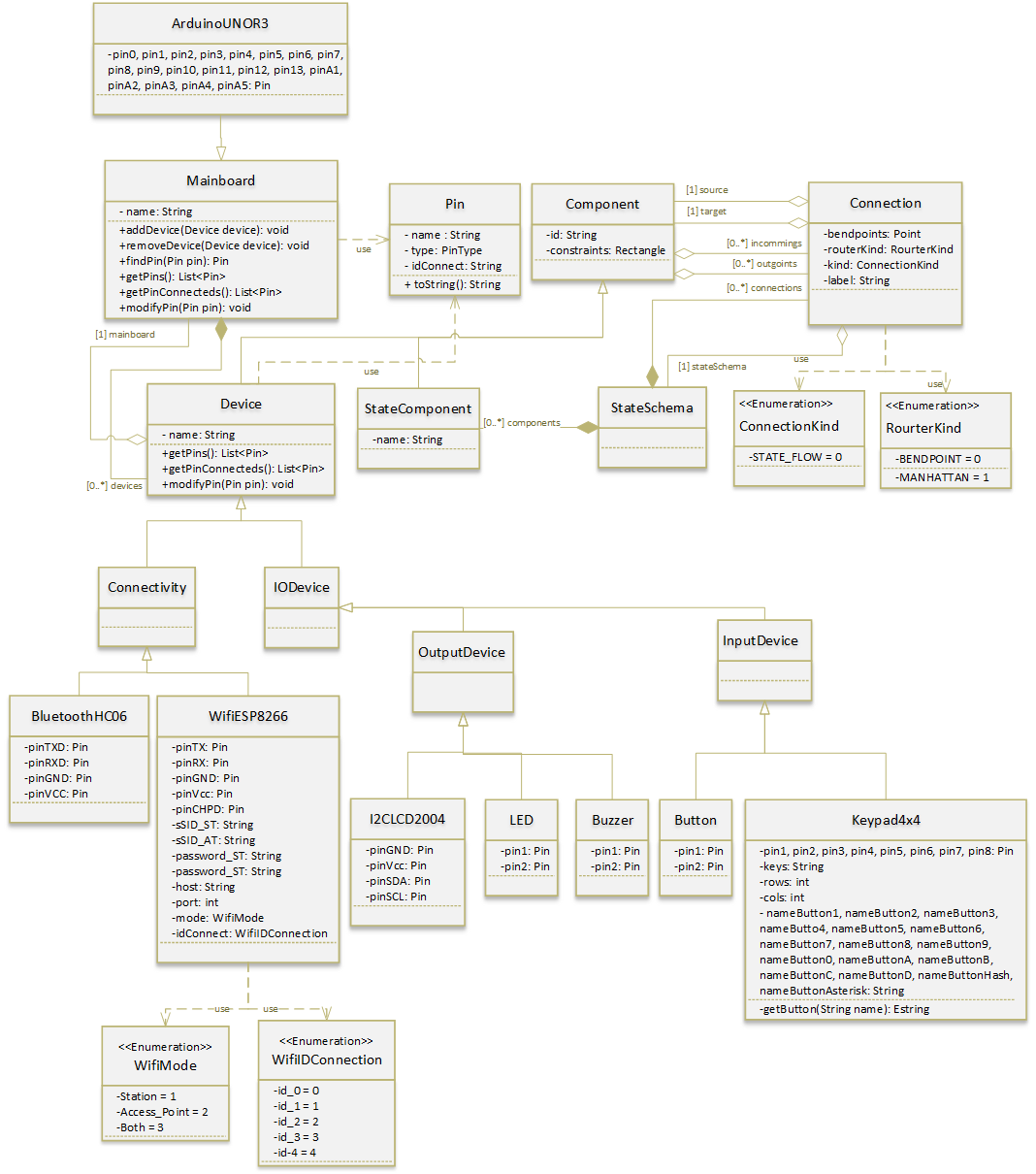
### Giao diện mô tả ứng dụng.

Trong giao diện này người dùng sẽ mô tả lược đồ trạng thái cho chương trình. Giao diện được thiết kế để có thể hỗ trợ để vẽ lược đồ trạng thái. Giao diện gồm các chức năng sau:

1. Thêm, xóa, sửa các đối tượng trong lược đồ trạng thái.
2. Hỗ trợ tính năng chỉnh sửa mô tả trạng thái trực tiếp.
3. Kiểm tra tính đúng đắn của lược đồ trạng thái trước khi phát sinh mã.
4. Hỗ trợ các chức năng cơ bản của giao diện đồ họa.

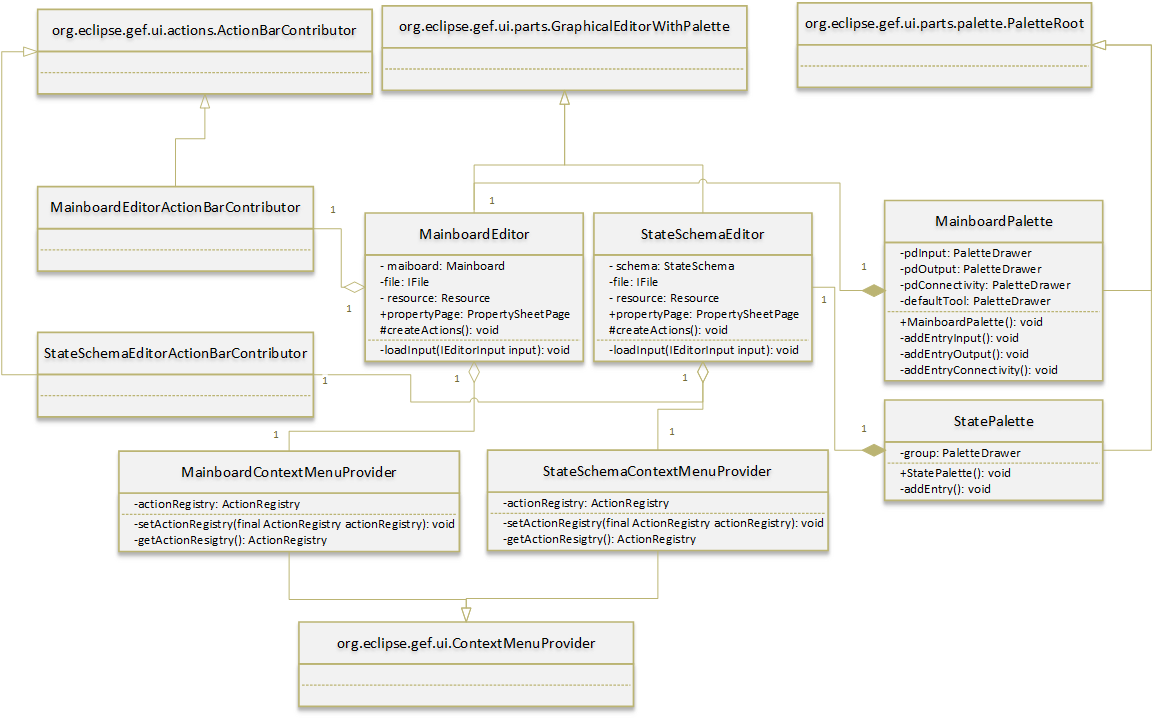
## Sơ đồ lớp Model

WAGEN làm việc với các thiết bị điện tử như bo mạch Arduino, Keypad3x3, LED… và thành phần của lược đồ trạng thái gồm StartPoint, Endpoint, StateFrame, Connection, Decision. Khi mô hình hóa, model sẽ mang những đặc điểm của đối tượng mà model mô tả. Đồng thời để có thể hiển thị, thao tác trên trình soạn thảo đồ họa, thì model cũng cần các đặc điểm của đối tượng đồ hòa.



Hình : Sơ đồ lớp model

## Thiết kế giao diện người dùng



Hình : Sơ đồ lớp Editor

Công cụ hỗ trợ việc mô tả thiết bị và mô tả lược đồ trạng thái cho ứng dụng. Vì vậy cần 2 giao diện đồ họa: Mainboard và StateSchema.

|  |  |
| --- | --- |
| Class | Chức năng |
| MainboardEditor | Tạo giao diện Mainboard để thiết đặt các thông tin thiết bị |
| StateSchemaEditor | Tạo giao diện StateSchema để mô tả lược đồ trạng thái |
| MainboardPalette | Tạo thanh Palette cho giao diện Mainboard |
| StatePalette | Tạo thanh Palette cho giao diện StateSchema |
| MainboardContextMenu | Tạo ContextMenu cho giao diện Mainboard |

Bảng 3‑2: Mô tả chức năng các lớp Editor

## Phân tích cú pháp

Trong các trạng thái có các dòng mô tả trạng thái dưới dạng văn bản. Mỗi một dòng thể hiện trạng thái của một thiết bị cụ thể. Vì vậy, cần hệ thống cú pháp quy định việc mô tả này. Thông qua bộ phân tích cú pháp có thể xác định được dòng mô tả này mô tả cho thiết bị nào.

Trên bo mạch có những loại có thể gắn nhiều thiết bị cùng loại. Nên việc mô tả cần xác định id. Ví dụ: trên bo mạch có thể có nhiều đèn LED. Để mô tả cho việc đèn LED bật. Sử dụng cú pháp: Display <id>, với “Display” là từ chỉ hành động, “<id>” là định danh để xác định thiết bị được thực hiện hành đông “Display” là thiết bị nào.

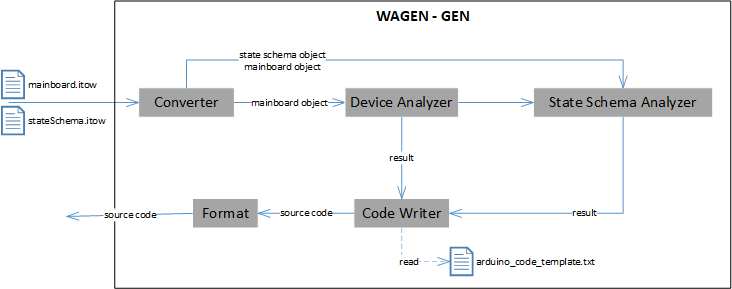
Nhưng cũng có loại chỉ được gắn 1 thiết bị. Nên việc mô tả không cần xác định id. Ví dụ: trên bo mạch chỉ sử dụng 1 LCD nên mô tả sẽ không cần id cho LCD.

Ngoài ra còn những mô tả để ám chỉ công việc phức tạp: create an order.

Vì thế cần có bộ phân tích cú pháp để xác định từng loại mô tả và bảng cú pháp cho các loại thiết bị.

## Bộ phát sinh mã

### Tổng quan kiến trúc



Hình : Sơ đồ phát sinh mã

Bộ phát sinh mã đảm nhận việc chuyển đổi các thiết kế và mô tả của người dùng, để phát sinh mã nguồn cho ứng dụng chạy trên thiết bị wearable.

Đầu vào là dữ liệu từ hai tệp nguồn iotw. Một tệp mô tả cho các thành phần cho bo mạch, một tệp mô tả cho hoạt động logic của ứng dụng bằng lược đồ trạng thái. Kết quả là chuỗi mã nguồn. Quá trình phát sinh mã được tiến hành qua 5 giai đoạn sau:

Giai đoạn 1: Converter - Chuyển đổi tệp nguồn dưới dạng văn bản thành đối tượng model tương ứng trong Java.

Giai đoạn 2: Device Analyzer - Phân tích thiết bị (Device) trên đối tượng bo mạch (Mainboard) để phát sinh mã dạng: khai báo tham chiếu (include), khai báo định nghĩa thiết bị và các phương thức đi kèm.

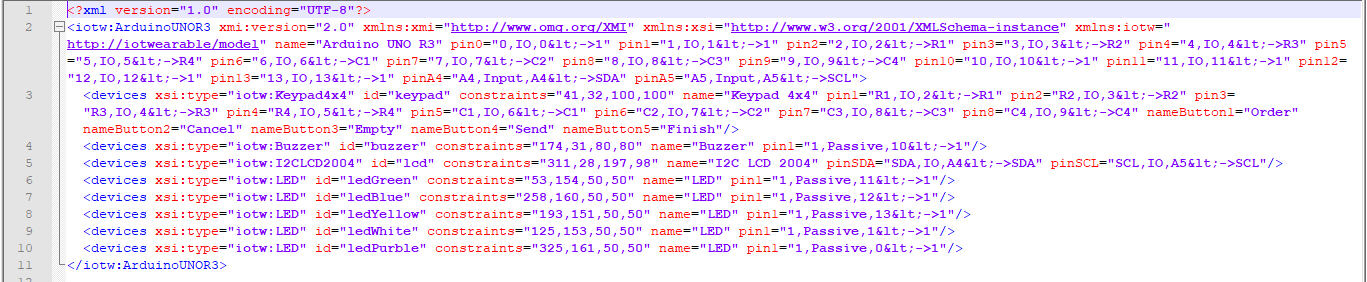
Giai đoạn 3: State Schema Analyzer – Phần tích lược đồ trạng thái để phát sinh mã theo logic trạng thái. Trong phần này bao gồm cả việc phân tích ngữ nghĩa. Phân tích ngữ nghĩa phải trả lời các câu hỏi sau: Mô tả cho thiết bị nào? Mô tả cho hành động gì?

Giai đoạn 4: Format – Định dạng cho mã nguồn.

### Giai đoạn 1: Converter - Chuyển đổi tệp nguồn.

Chuyển đổi dữ liệu dạng văn bản của tệp .iotw (XML) thành đối tượng model trên ngôn ngữ Java. Việc này giúp truy xuất và truyền dữ liệu đến các thành phần khác trong quá trình phát sinh mã được dễ dàng.

Tệp nguồn mô tả cho bo mạch Arduino UNO R3



Hình : Tệp mainboard.iotw

Tệp nguồn mô tả cho lược đồ trạng thái.

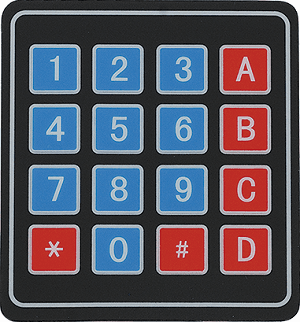


Hình : Tệp stateSchema.iotw

Việc chuyển đổi sử dụng Resource của EMF, vì thế đảm bảo tính đúng đắn của đối tượng khi được chuyển đổi.

### Giai đoạn 2: Device Analyzer - Phân tích các thiết bị

Mỗi thiết bị, tùy vào độ phức tạp, sẽ không có hoặc có các thư viện riêng để hoạt động trên bo mạch Arduino. Ví dụ đối với loại Keypad4x4



Hình : Keypad4x4

Để hoạt động trên bo mạch Arduino cần sử dụng thư viện hỗ trợ là <https://github.com/Chris--A/Keypad>. Mỗi thư viện được tham chiếu trong mã nguồn theo cú pháp #include <tên thư viện>.

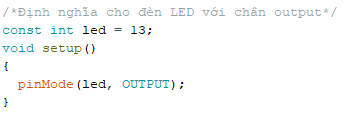


Hình : Ví dụ về khai báo thư viện cho keypad

Còn đối với đèn LED thì không cần thư viện, vì vậy việc đầu tiên của Device Analyzer phải xác định được thư viện tương ứng cho mỗi thiết bị và phát sinh mã khác báo theo cú pháp trên.

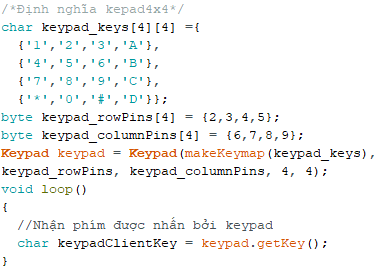
Sau khi xác định được thư viện tham chiếu, việc cần thực hiện tiếp theo là xác định thiết bị được kết nối với bo mạch Arduino như thế nào? Để làm điều này cần dựa vào đặc điểm của từng loại thiết bị và các chân được kết nối đến bo mạch Arduino.

Đối với đèn LED với 2 chân cắm và một trong 2 chân khi được gắn với bo mạch chính sẽ làm nhiệm vụ nhận các tính hiệu từ bo mạch. Để xác định chân nào dùng cho việc này trong mã cho Arduino như sau:



Hình : Ví dụ mã định nghĩa cho đèn LED trong Arduino

Đối với keypad cụ thể là Keypad4x4 sử dụng tổng cộng 8 chân để kết nối đến bo mạch và cần được cài đặt để có thể truyền tín hiệu vào bo mạch. Mã được thực hiện như sau:



Hình : Ví dụ mã định nghĩa cho keypad4x4 trong Arduino

Nhưng vậy đối với mỗi loại thiết bị, đều có những đoạn mã khai báo, định nghĩa thiết bị và những chức năng (hàm) đặc trưng cho mỗi loại. Những đoạn mã này có thể khái quát để tạo ra những mẫu khai báo (template) cho mỗi loại thiết bị tương ứng như trên.

Sơ đồ lớp phân tích thiết bị



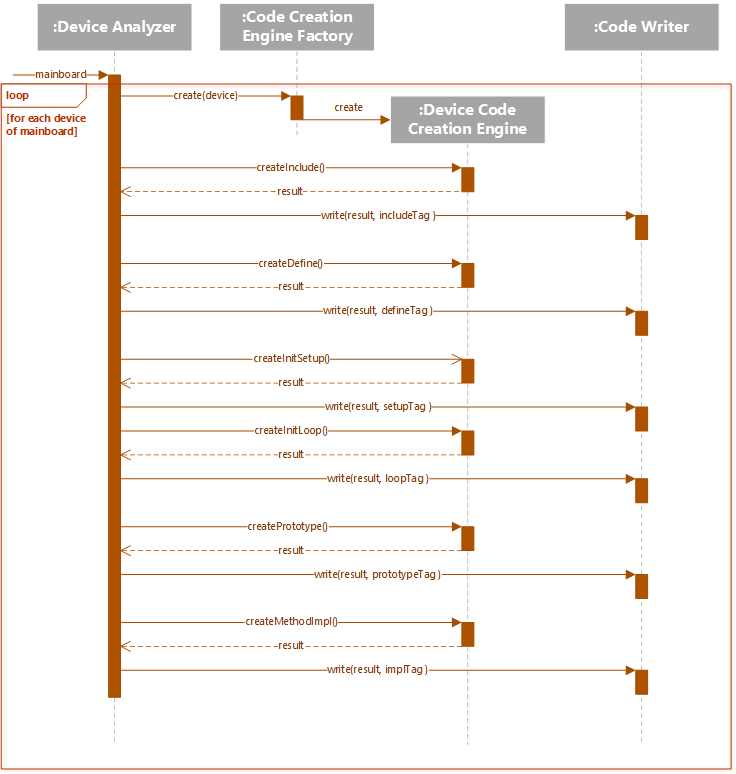
Hình : Sơ đồ lớp Device Analyzer

Đây là nơi tạo ra các đoạn mã cho các thiết bị. Cụ thể với các phương thức sau:

|  |  |
| --- | --- |
| Tên phương thức | Chức năng |
| createInclude() | Tạo mã khai báo tham chiếu. |
| createDefine() | Tạo mã định nghĩa thiết bị. |
| createInitSetup() | Tạo mã thiết lập thiết bị cho bo mạch. |
| createInitLoop() | Tạo mã cho các chức năng đặc trưng cho mỗi loại thiết bị. |
| createPrototype() | Tạo mã khai báo prototype cho những hàm đặc biệt của thiết bị mà được WAGEN hỗ trợ. |
| createMethodImp() | Tạo mã hàm đặc biệt của thiết bị mà được WAGEN hỗ trợ. |
| createFormCommand(String syntax, LinkedList<Token>) | Tạo mã từ dòng đặc tả theo cú pháp trong bộ cú pháp của WAGEN. |

Bảng 3‑3: Các phương thức tạo mã cho thiết bị

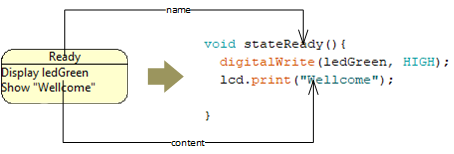
Sơ đồ trình tự phân tích thiết bị



Hình : Sơ đồ trình tự phân tích thiết bị

### Giai đoạn 3: State Schema Analyzer – Phần tích lược đồ trạng thái

#### Tạo mã với khung trạng thái



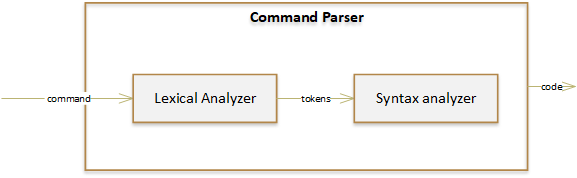
Hình : Tạo mã với khung trạng thái

Tên của khung trạng thái, được chuyển thành tên của hàm bằng cách thêm tiền tố “state” trước tên khung trạng thái. Với nội dung đặc tả là mã trong thần hàm.

#### Bộ phân tích cú pháp lệnh

Các mô tả đã được xác định cấu trúc – DefinedCommand. Đây là những đoạn mã tuần theo cú pháp.

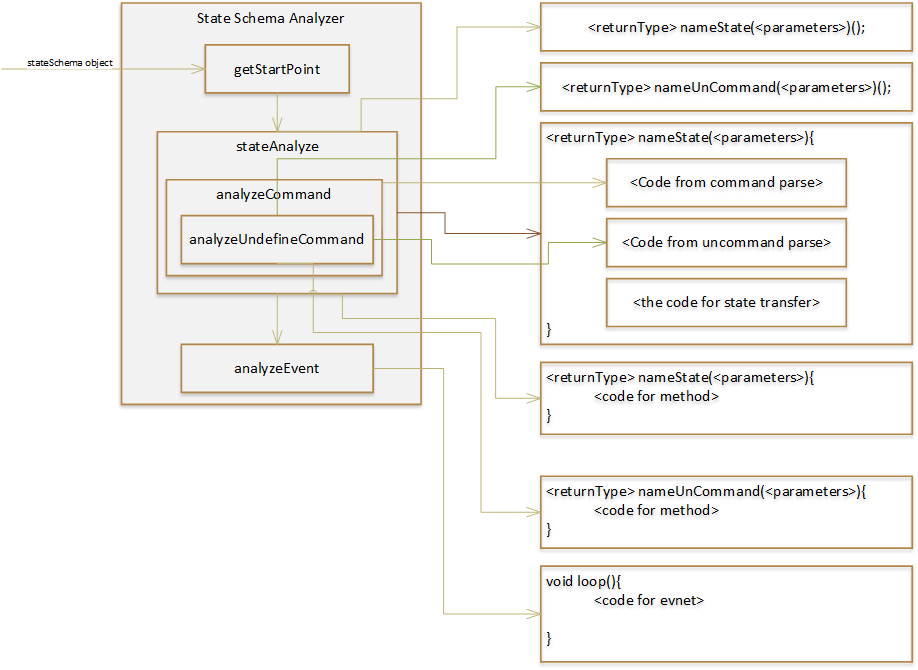
Các mô tả chưa xác định cấu trúc - UndifinedCommand. Đây là loại mô tả như một câu văn bình thường. Ví dụ như “Create an order” hay “Send request”.



Hình : Bộ phân tích cú pháp lệnh

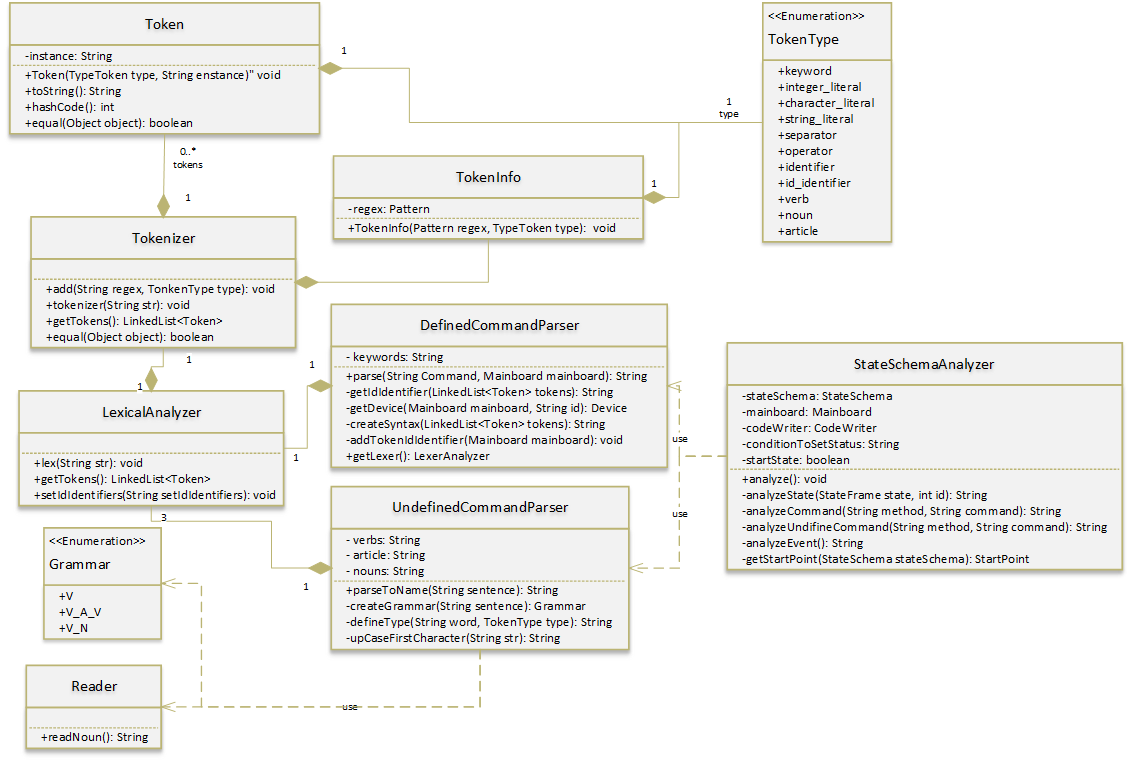
Lexical Analyzer sẽ nhận dạng các Token của câu lệnh (command). Từ đó đưa ra danh sách các token của câu lệnh. Danh sách token này sẽ được sử dụng cho quá trình phân tích cú pháp.

#### Cơ chế tạo mã



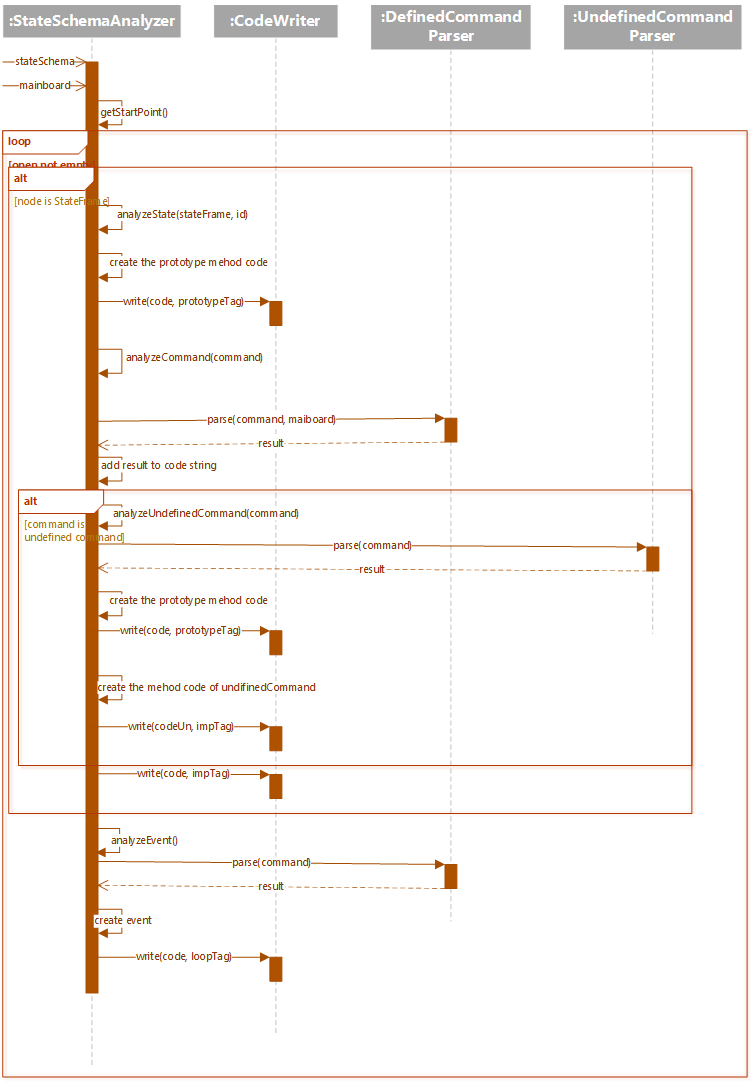
Hình : Sơ đồ ánh xạ các giai đoạn phát sinh mã của lược đồ trạng thái

#### Sơ đồ lớp phân tích lược đồ trạng thái



Hình : Sơ đồ lớp phân tích lược đồ trạng thái

#### Sơ đồ tuần tự phân tích lược đồ trạng thái



Hình : Sơ đồ trình tự phân tích lược đồ trạng thái

### Giai đoạn 4: Format – Định dạng mã nguồn

Mã nguồn sau khi được phát sinh, phải được kiểm chứng tính đúng đắn, phát triển và bảo trì. Vì vậy mã nguồn được tạo ra cần trình bày một cách khoa học.

Format sẽ định dạng mã nguồn, trình bày các dòng mã theo định dạng ngôn ngữ C.

## Tạo tài liệu hướng dẫn lắp đặt và danh sách thiết bị

Ngoài việc phát sinh mã nguồn cho ứng dụng được mô tả. Ứng dụng còn phát sinh tài liệu hướng dẫn lắp đặt và danh sách các thiết bị cho hệ thống.

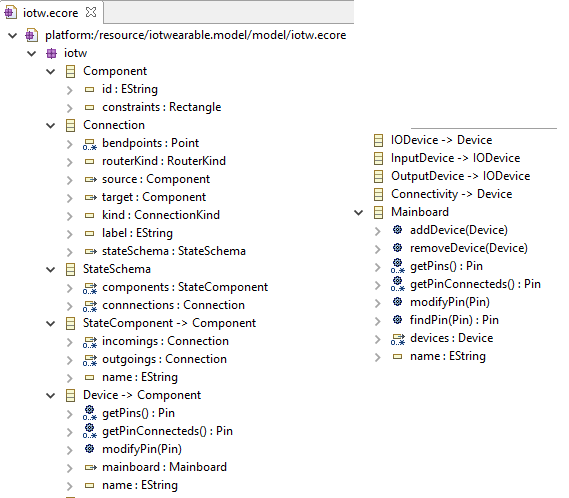
Tài liệu gồm 2 phần:

1. Danh sách thiết bị cần dùng cho ứng dụng trên.
2. Bảng hướng dẫn lắp đặt thiết bị.

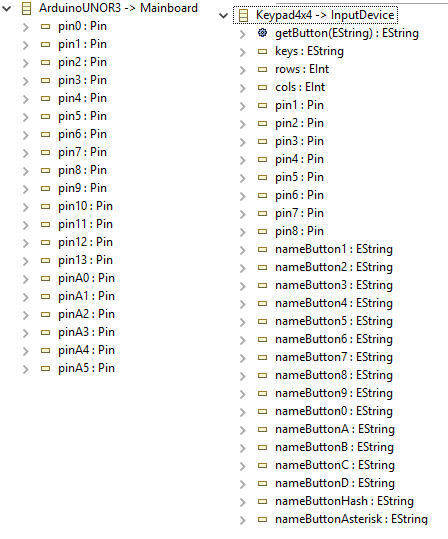
# : HIỆN THỰC

## Thiết kế model với EMF

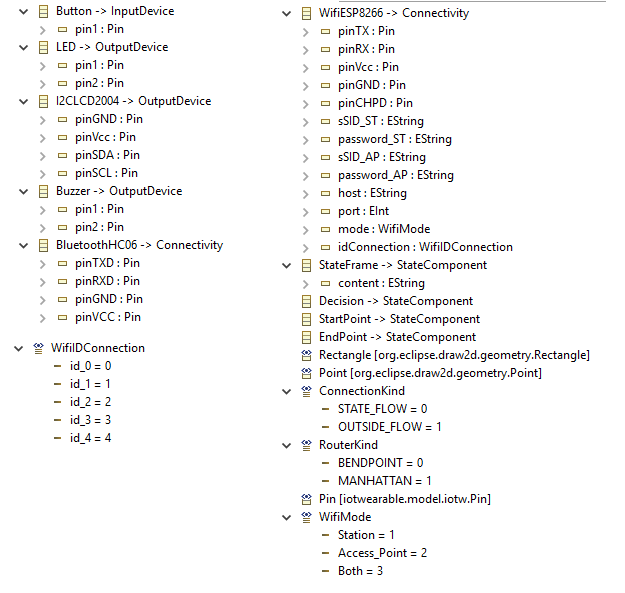
Tạo model sử dụng EMF



Hình : Thiết kế model sử dung EMF-1



Hình : Thiết kế model sử dung EMF-2

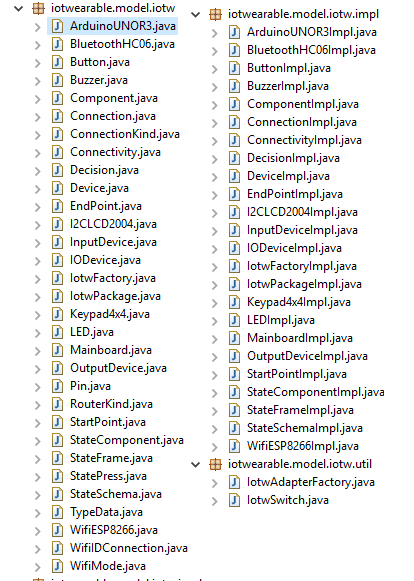


Hình : Thiết kế model sử dung EMF-3

## Phát sinh model với EMF

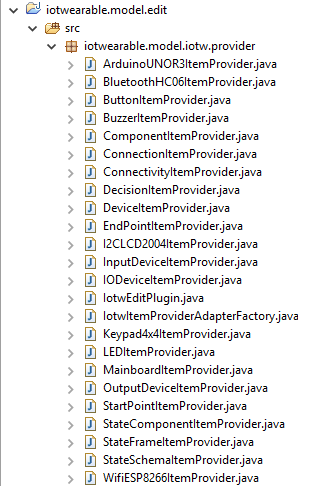
Sau khi thiết kế model tiến hành phát sinh các plug-in

### model plug-in



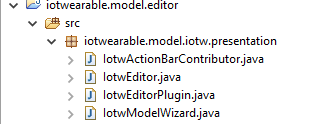
Hình : model plug-in

### model.edit plug-in



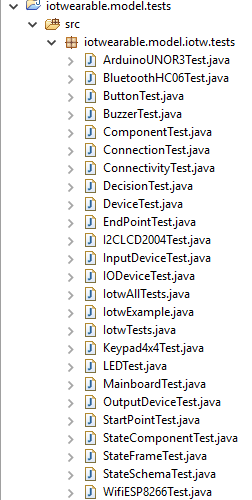
Hình : model.edit plug-in

### model.editor plug-in



Hình : model.editor plug-in

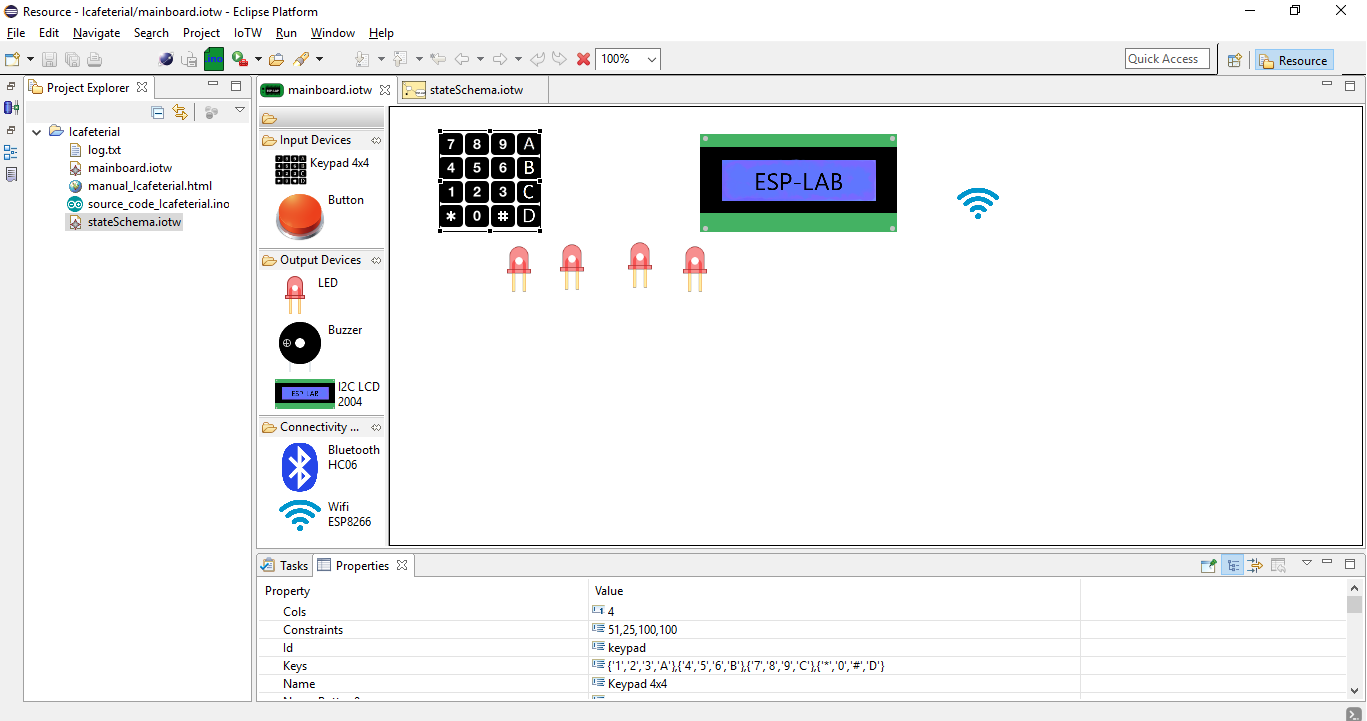
### model.test plug-in



Hình : model.test plug-in

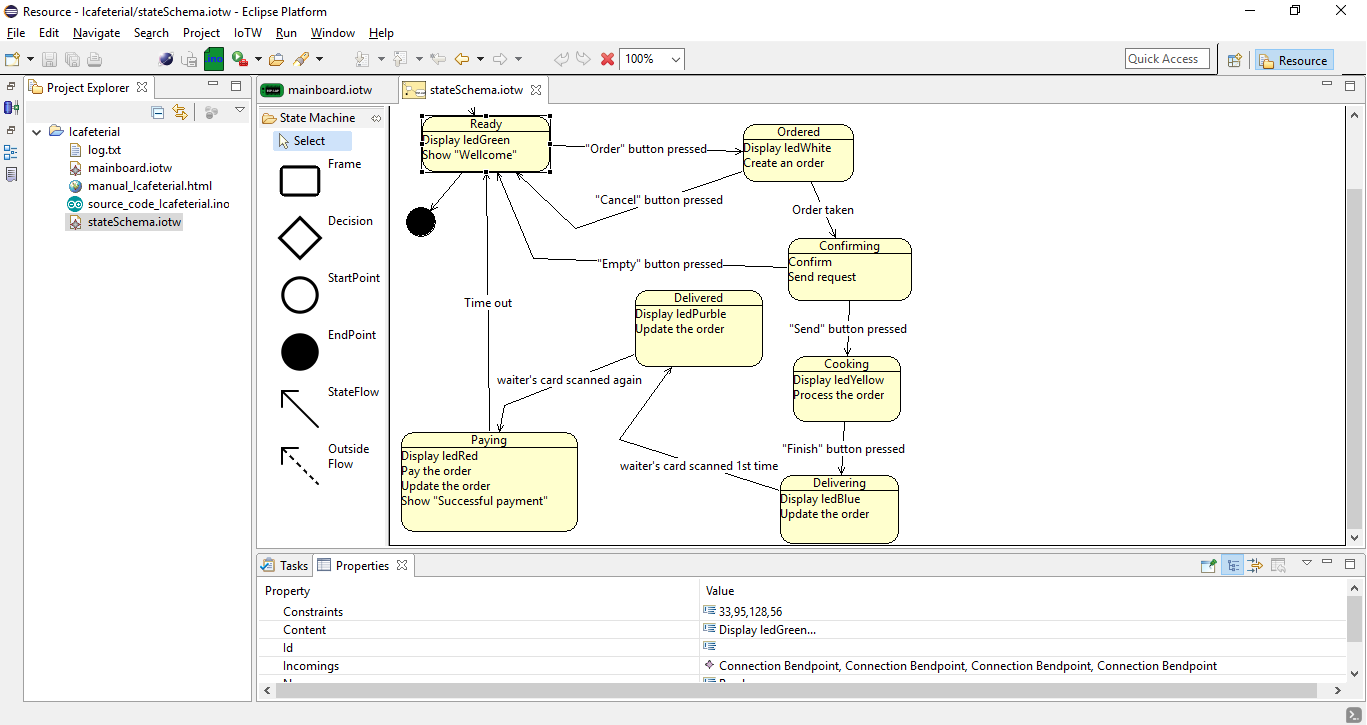
## Xây dựng giao diện người dùng với GEF

### Giao diện cho bo mạch và thiết bị



Hình : Giao diện cho bo mạch và thiết bị

### Giao diện cho máy trạng thái



Hình : Giao diện cho máy trạng thái

## Xây dựng bộ cú pháp

|  |  |  |  |
| --- | --- | --- | --- |
| **Cú pháp** | **Mô tả** | **Thiết bị** | **Ví dụ** |
| Display <id> | Kích hoạt một thiết bị. | LED | Display ledRed |
| Hidden <id> | Ngừng hoạt động cho một thiết bị | LED | Hidden ledRed |
| Blink <id> | Điều khiển thiết bị nhấp nháy | LED, I2CLCD | Blink lelRed, Blink lcd |
| Beep <id> | Phát ra tiếng kêu Beep | Buzzer | Beep buzzer |
| Show <String> | Hiển thị chuỗi lên thiết bị hiển thị | LCD | Show "Xin chao" |
| <String> button pressed | Mô tả sự kiện nhấn phím | keypad | "Cancel" button pressed |
| <id> push | Nhận sự kiện nhấn 1 push button | push button | button push |

Bảng 4‑1: Các từ ngữ ánh xạ để mô tà lược đồ trạng thái

## Kiểm chứng

Chương trình điều khiển 3 đèn LED bật tắt bằng Keypad. Ở trạng thái ban đầu 3 đèn LED đều tắt. Khi nhấn button 0 trên keypad đèn led1 sẽ sáng. Sau đó nhấn button 1 đèn led2 sẽ sáng, led1 sẽ tắt. Sau đó nhấn button 0 cả 3 đèn sẽ sáng. Sau đó nhấn button 2 tất cả đèn sẽ tắt.

### Bước 1: Tạo dự án

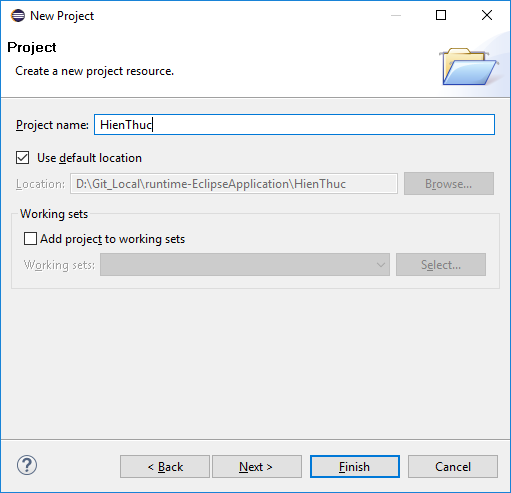
Trong dự án phát sinh mã cần 2 file: file mô tả thiết bị trên bo mạch và file đặc tả lược đồ trạng thái.

Chon menu File 🡪 New 🡪 project… Cửa sổ tiếp theo hiện ra, nhấn vào thư mục có tên General 🡪 Project 🡪 Next như hình sau



Hình : Tạo project

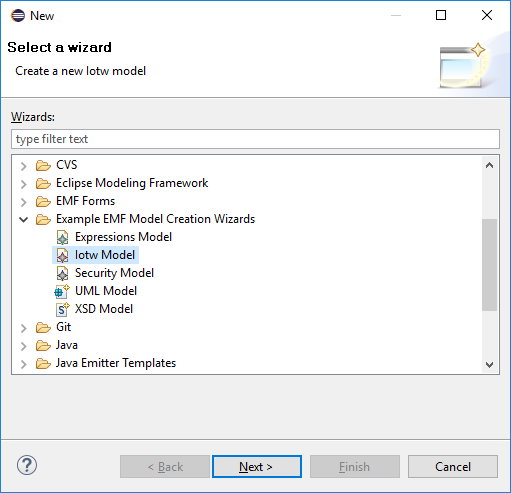
Màn hình kế tiếp trong như thế này. Nhập tên project vào mục Project Name 🡪 Finish



Hình : Đặt tên project

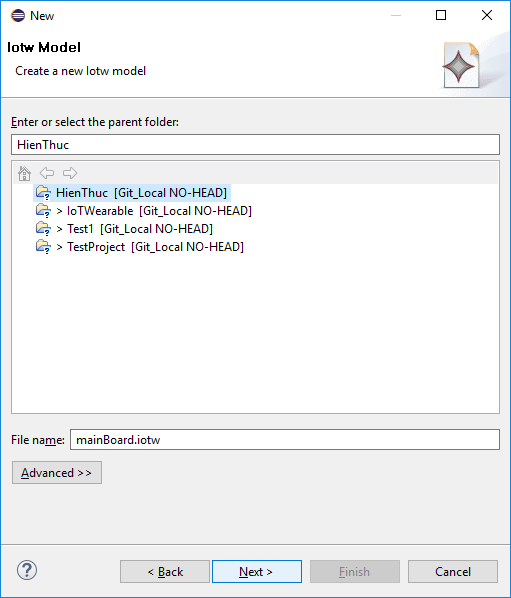
#### Tạo bo mạch.

Bước 1: Click phải chuột vào project vừa tạo 🡪 New 🡪 Other… Giao diện Select a Wizard hiện ra, chọn Example EMF Model Creation Wizards a, trong các thành phần xổ ra sau đó, chọn Iotw Model 🡪Next.



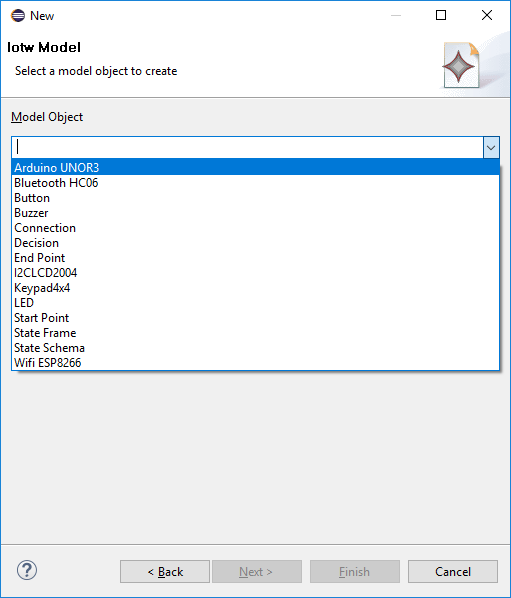
Hình : Tạo bo mạch cho dự án

Bước 2: Màn hình kế tiếp trong như thế này. Nhập tên của bo mạch vào mục File Name 🡪 Next.



Hình : Đặt tên cho bo mạch

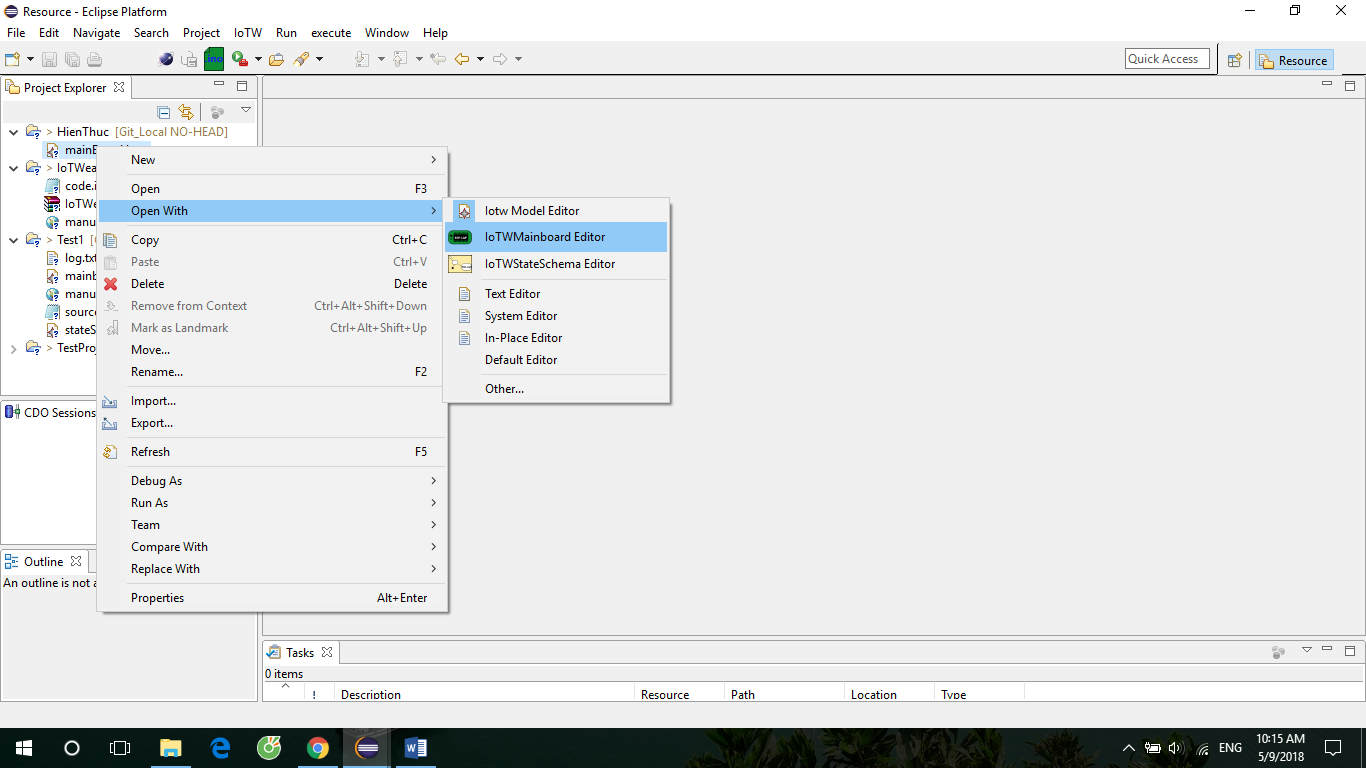
Bước 3: Màn hình kế tiếp sẽ hiện ra trong mục Model Object chọn loại bo mạch như hình sau. Chọn loại Arduino muốn sử dụng 🡪 Finish.



Hình : Chọn loại Arduino cho bo mạch

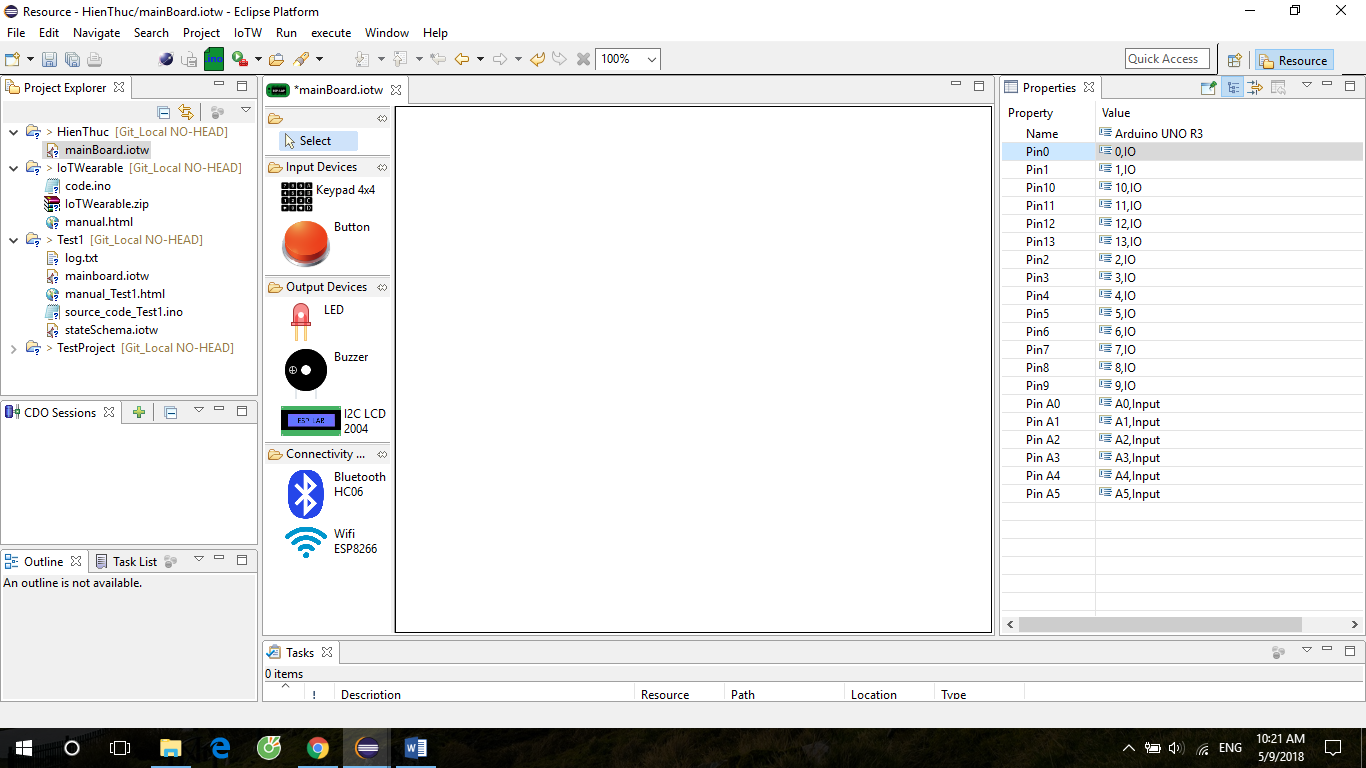
Bước 4: Mở bo mạch với giao diện cấu hình thiết bị

Click phải chuột vào File vừa tạo 🡪 Open With 🡪 IoTMainboard Editor



Hình : Mở giao diện cấu hình thiết bị

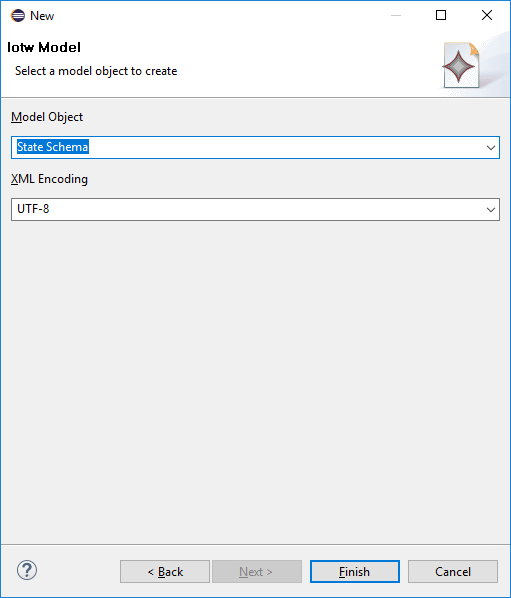
Giao diện mô tả hình thiết bị.



Hình : Giao diện cấu hình thiết bị

#### Tạo lược đồ trạng thái

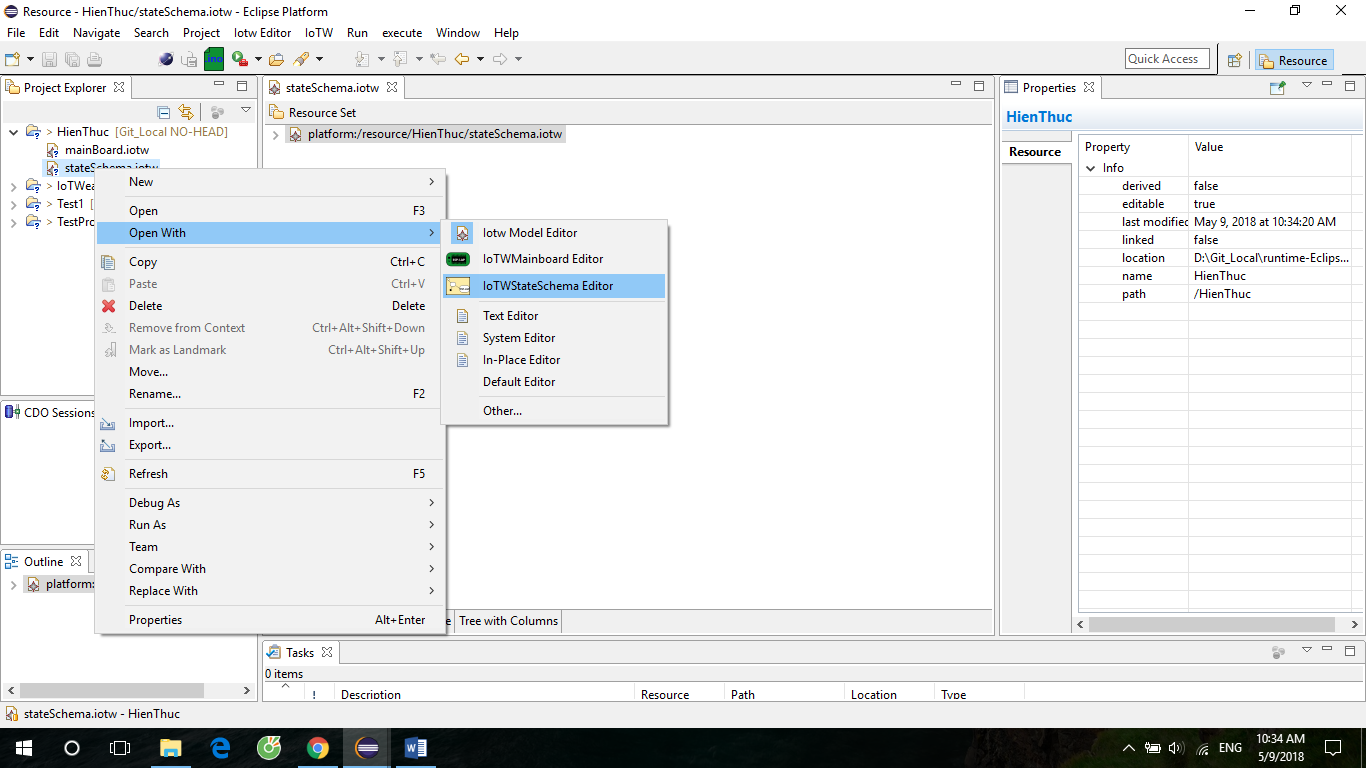
Bước 1: thực hiện tương tự tạo bo mạch. Nhưng đến bước 3 mục Object Model chọn StateSchema.



Hình : Chọn loại Object cho lược đồ trạng thái

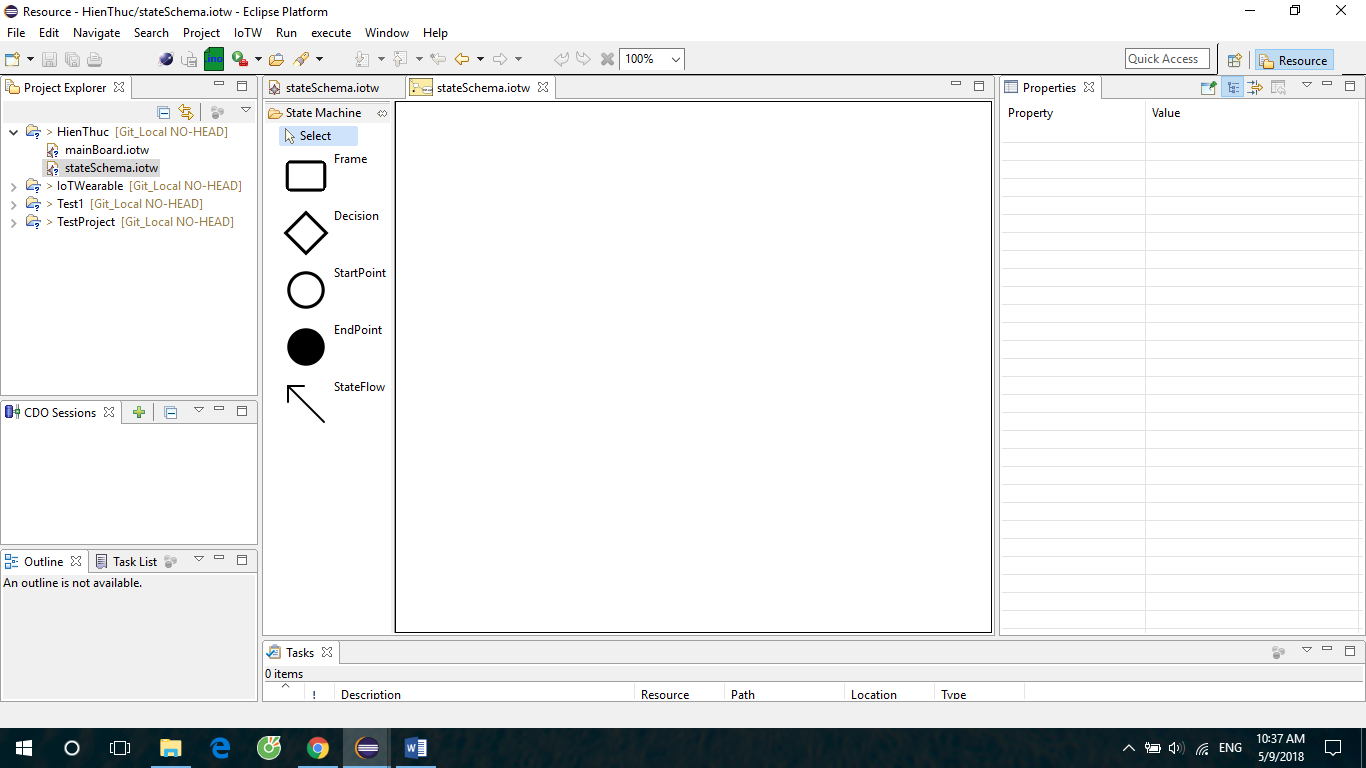
Bước 2: Mở giao diện mô tả lược đồ trạng thái

Click phải chuột vào File StateSchema vừa tạo 🡪 Open With 🡪 IoTStateSchema Editor.



Hình : Mở giao diện mô tả lược đồ trạng thái

Giao diện mô tả lược đồ trạng thái.

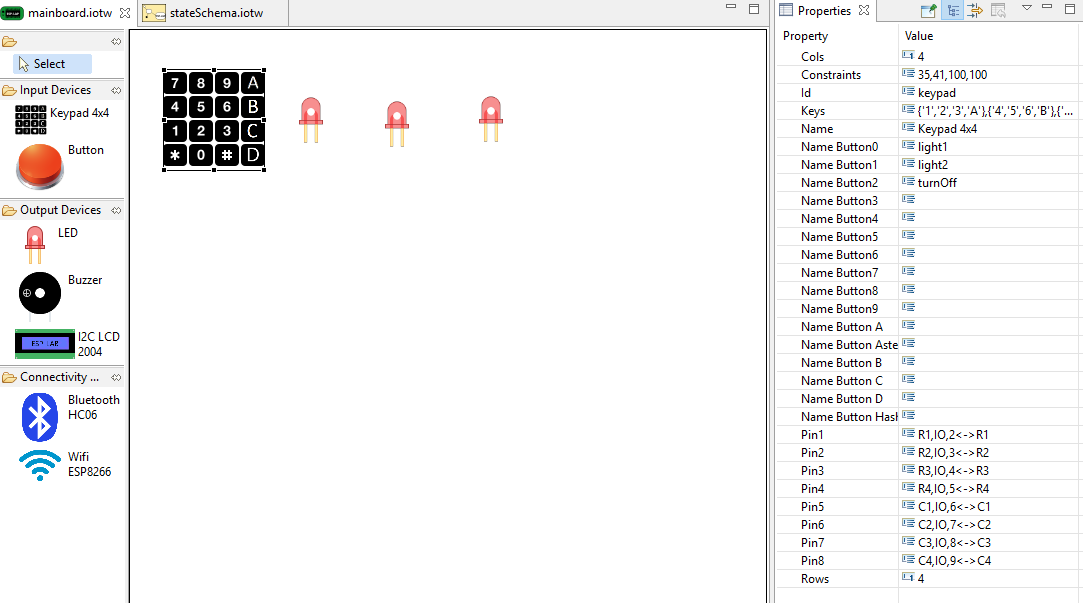


Hình : Giao diện mô tả lược đồ trạng thái.

### Bước 2: Thêm và cài đặ thông số cho các thiết bị.

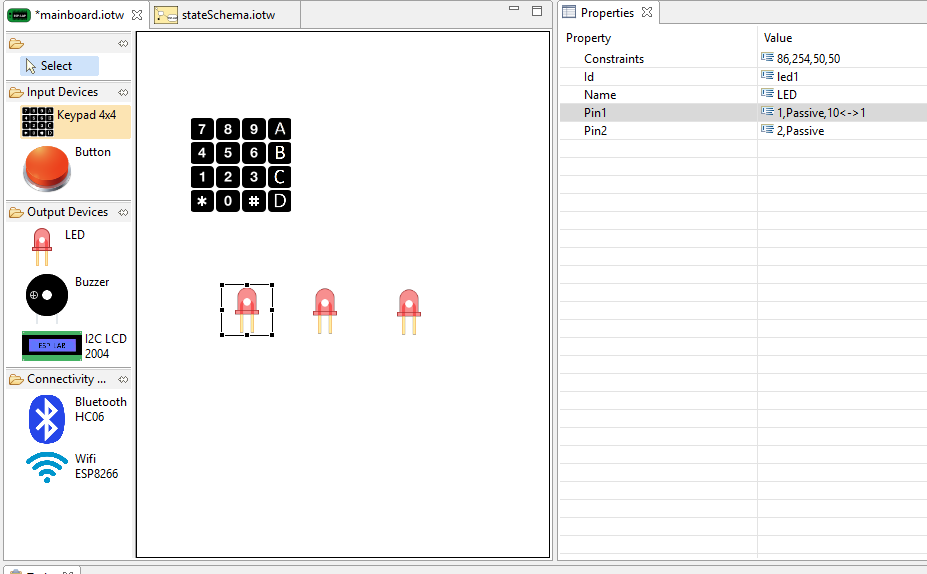
Kéo thả keypad4x4 và 3 đèn LED vào khung soạn thảo. Sau đó cài đặt các thông số cho mỗi thiết bị.

Cài đặt thông số cho keypad4x4:

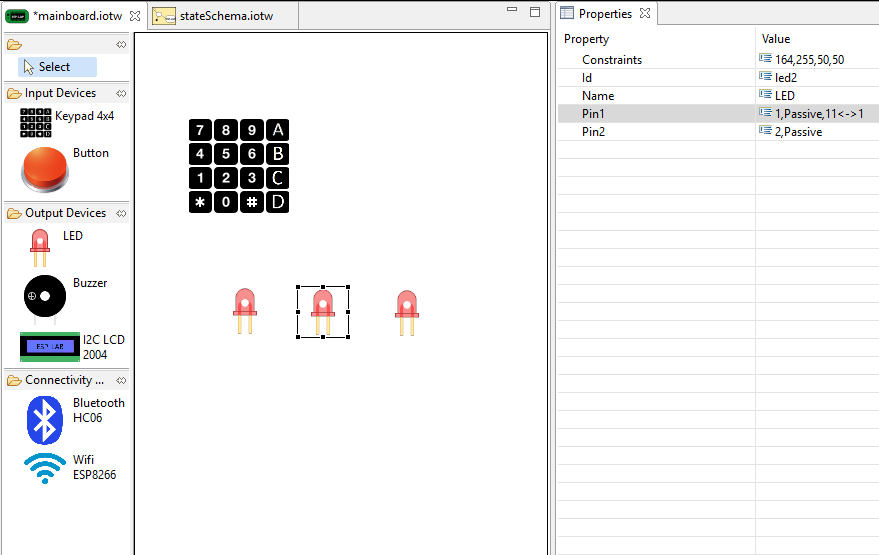


Hình : Cài đặt thông số cho Keypad 4x4

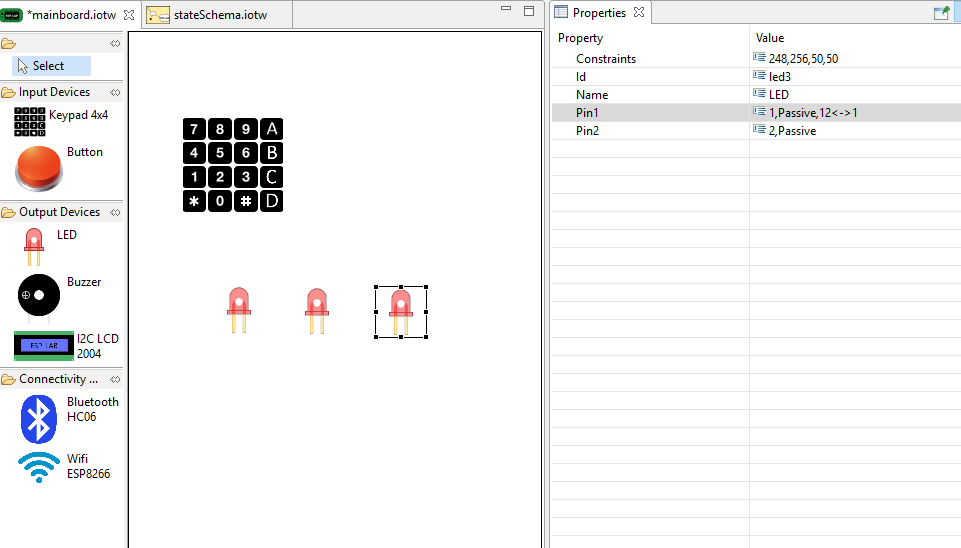
Cài đặt thông số cho các đèn LED



Hình : Cài đặt thông số cho đèn LED 1



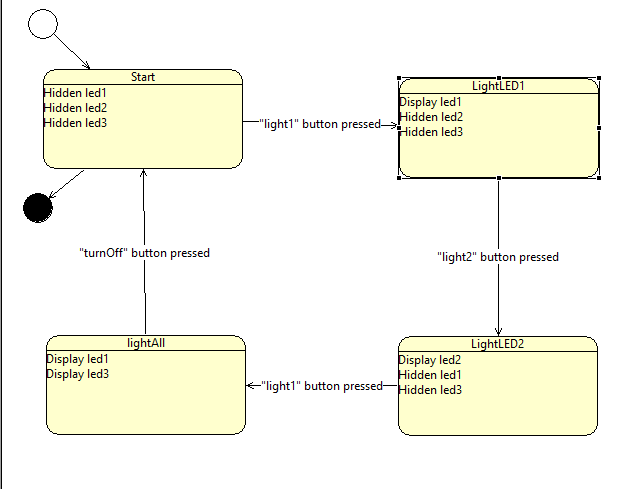
Hình : Cài đặt thông số cho đèn LED2



Hình : Cài đặt thông số cho đèn LED 3

### Bước 2: Mô tả lược đồ trạng thái chương trình.

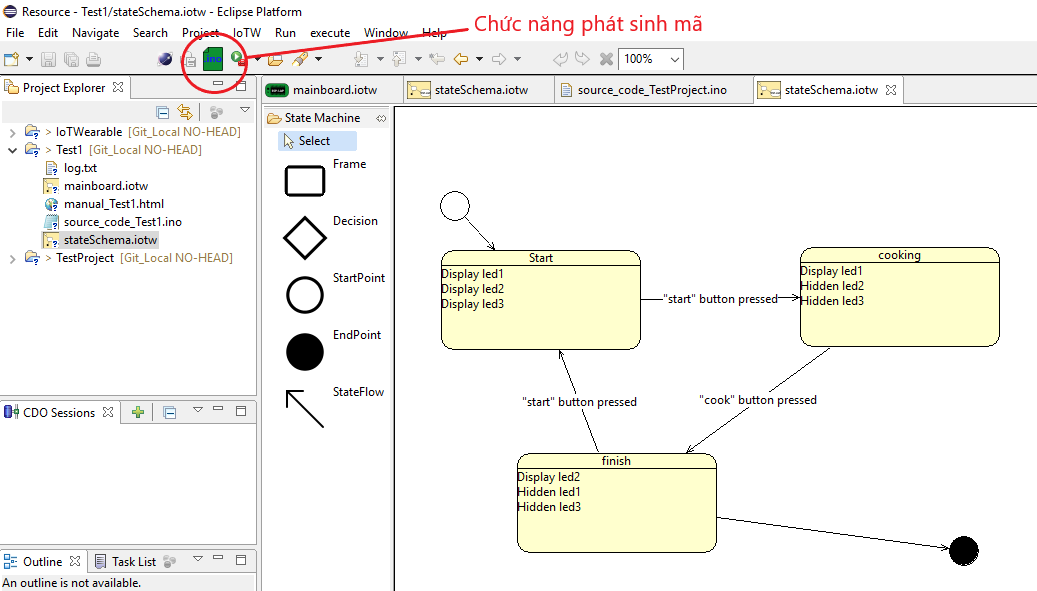
Sau khi mô tả các thiết bị cần thiết cho chương trình. Mô tả ứng dụng trong giao diện mô tả lược đồ trạng thái. Kéo thả các thành phần vào khung soạn thảo. Thay đổi nội dung để mô tả ứng dụng. Sau khi mô tả sẽ có hình như sau:



Hình : Lược đồ trạng thái cho ứng dụng điều khiển đèn LED

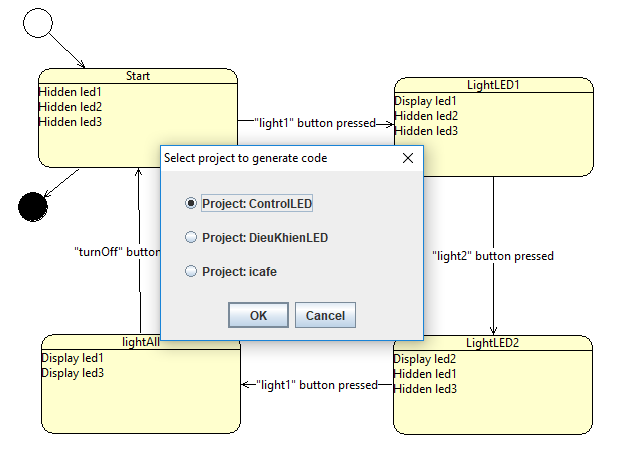
### Bước 3: Thực hiện chức năng phát sinh mã

Bước 1: Chọn chức năng phát sinh mã trên thanh công cụ.



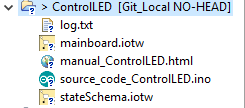
Hình : Chọn chứ năng phát sinh mã

Bước 2: Chọn dự án muốn phát sinh mã



Hình : Chọn dựa án

Kết quả sau khi phát sinh mã.

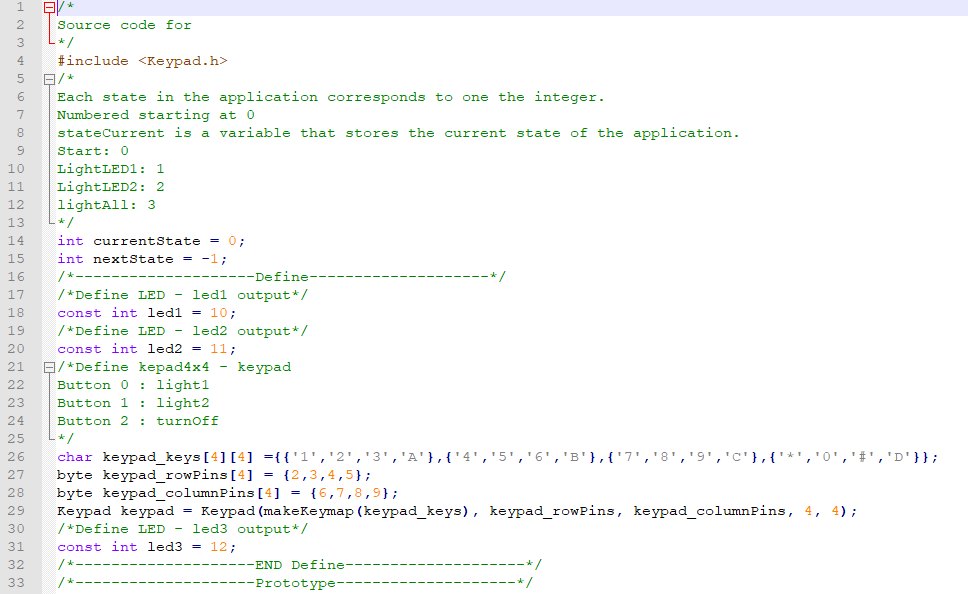


Hình : Kết quả sau khi tạo mã

|  |  |
| --- | --- |
| File | Chức năng |
| Log.txt | Ghi lại thông tin trong quá trình phát sinh mã. Bao gồm các lỗi nếu có. |
| Manual\_Test1.html | Tài liệu hướng dẫn của dự án vừa tạo. |
| Source\_code\_Test1.ino | File mã được phát sinh từ mô tả thiết bị và lược đồ trạng thái |

Bảng 4‑2: Thông tin các file được tạo ra khi phát sinh mã

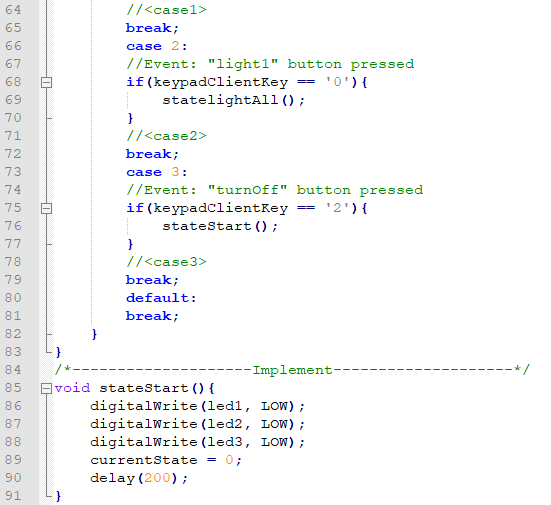
### Mã nguồn được phát sinh



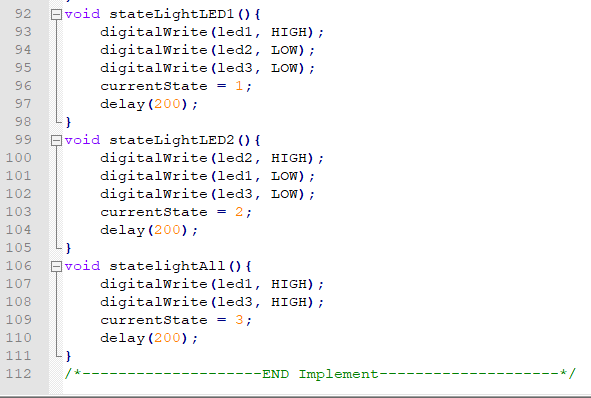
Hình : Mã nguồn được phát sinh phần 1



Hình : Mã nguồn được phát sinh phần 2

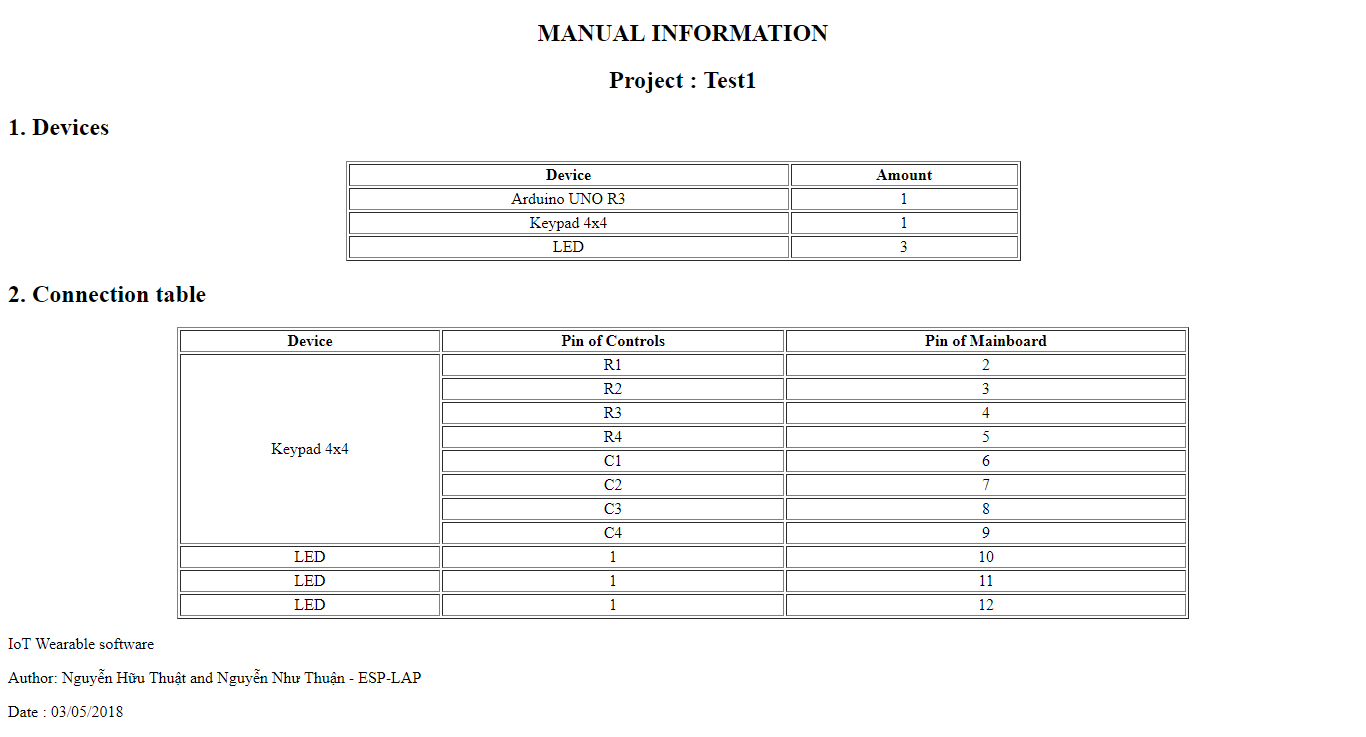


Hình : Mã nguồn sau khi phát sinh phần 3



Hình : Mã nguồn sau khi phát sinh phần 4

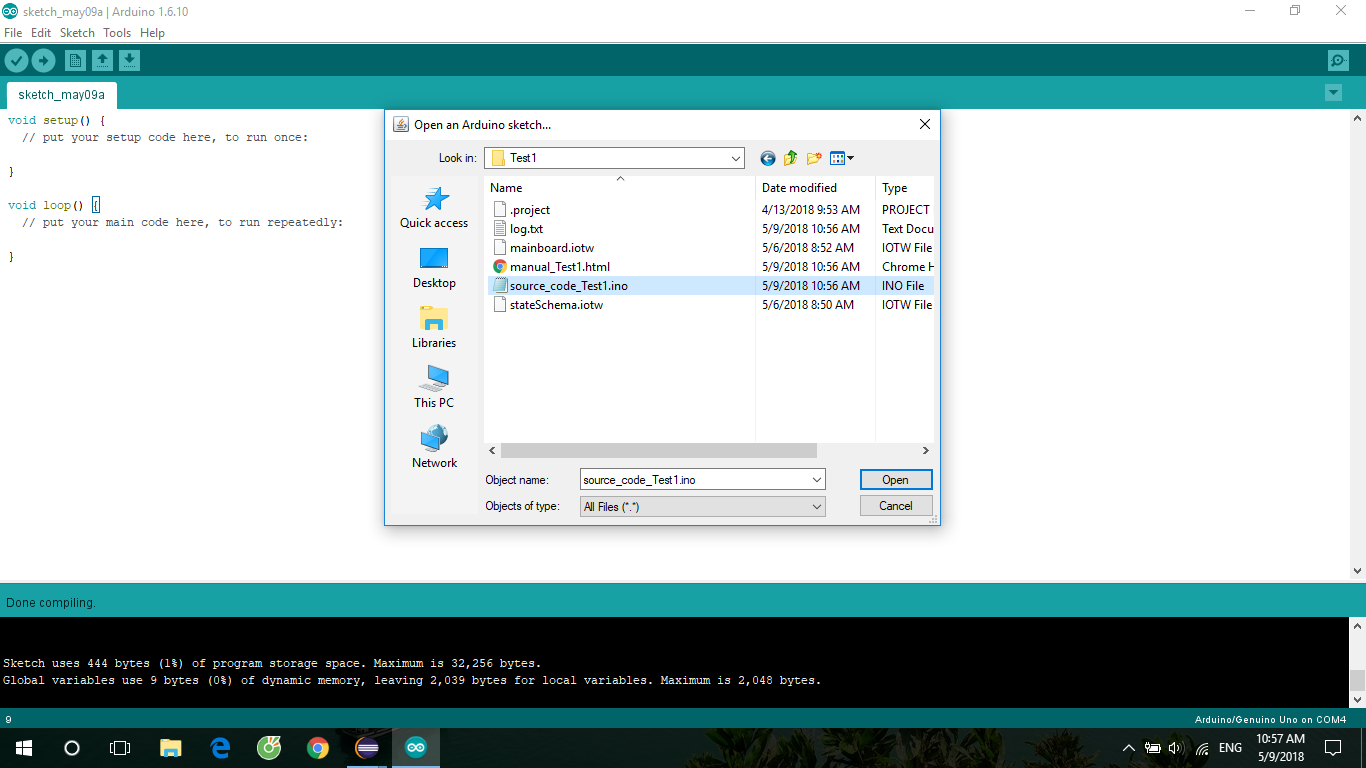
### Tài liệu hướng dẫn cho ứng dụng

****

Hình : Tài liệu cho ứng dụng mô tả

### Build file mã nguồn vào Arduino UNO R3

Bước 1: Mở chương trình Arduino IDE 🡪 File 🡪 Open 🡪 Chọn đường dẫn lưu file mã vừa phát sinh 🡪 Open. Như hình sau.

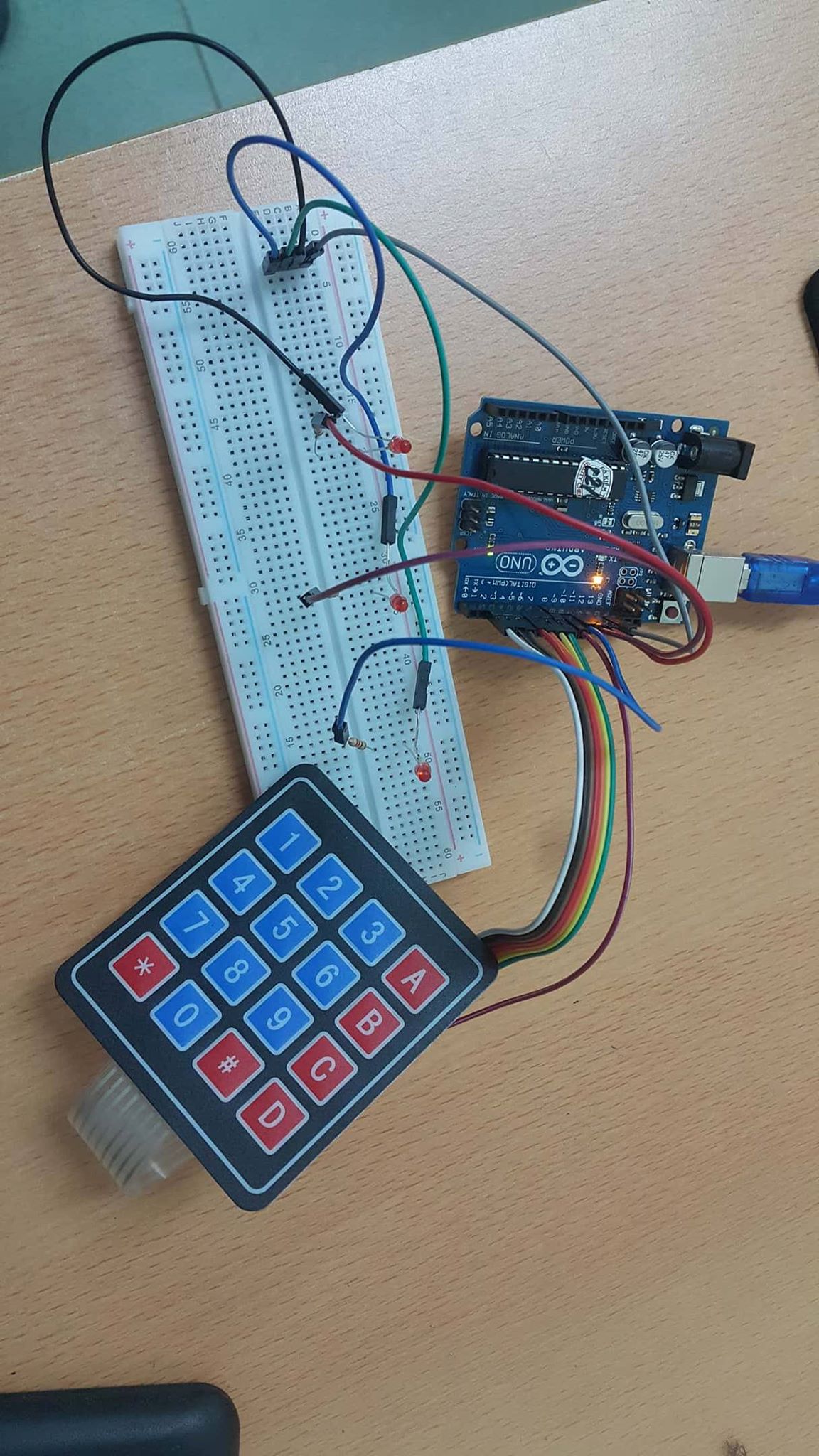


Hình : Mở File mã vừa phát sinh

Bước 2: Kết nối các thiết bị với bo mạch theo tài liệu hướng dẫn được phát sinh

Bước 3: Kết nối Arduino với Máy tính. Tải thư viện Keyad nếu Arduino IDE thiếu. Theo đường dẫn

Trạng thái ban đầu 3 đèn LED đều tắt



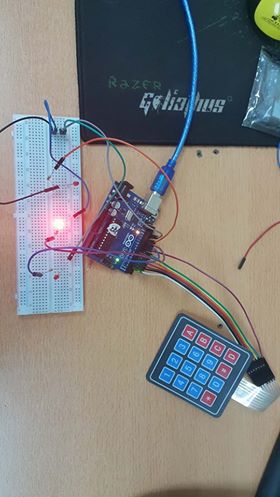
Hình : Kết nối thiết bị thực tế

Trạng thái khi nhấn nút số 0



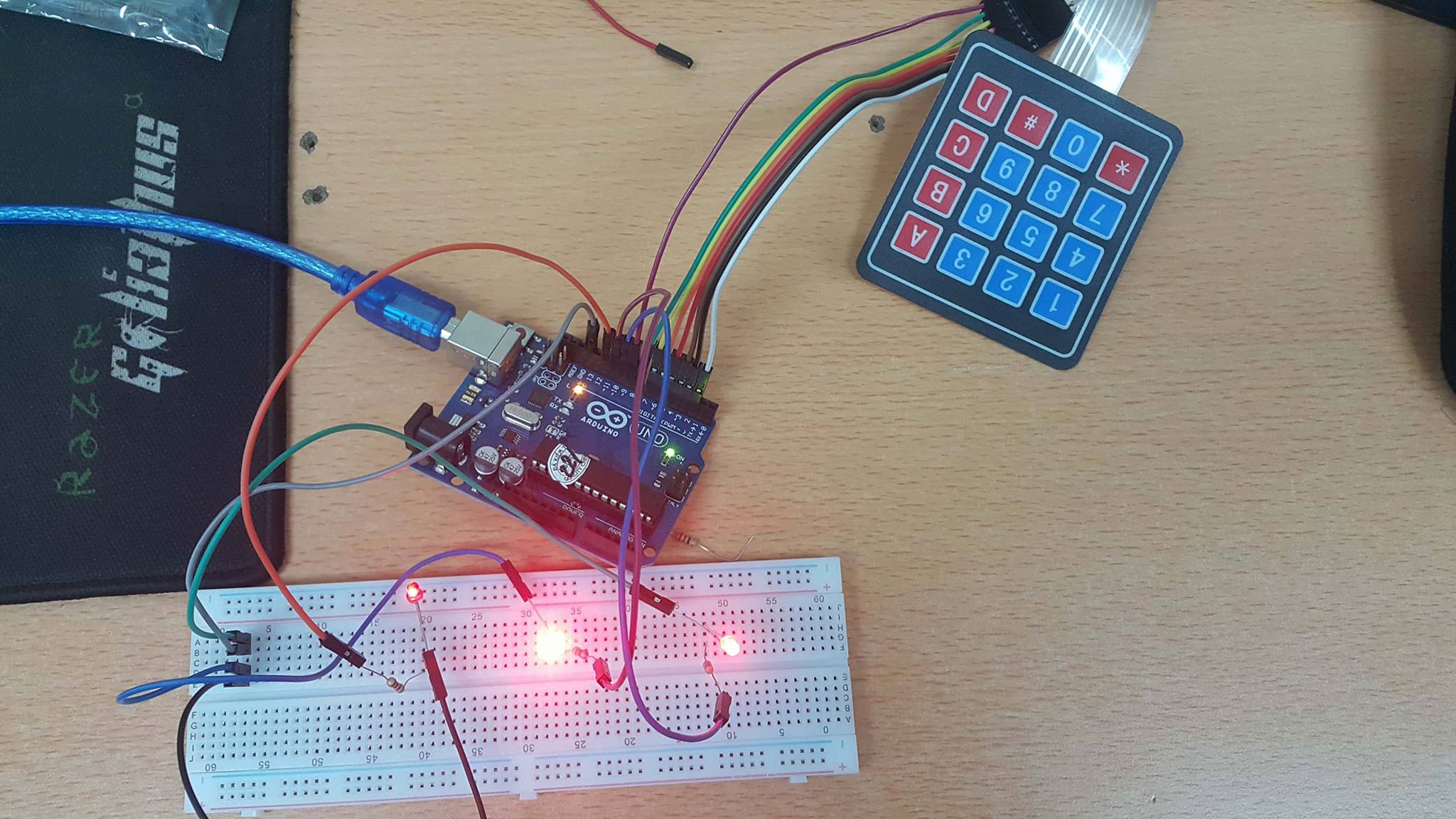
Hình : Trạng thái khi nhấn nút số 0

Trạng thái đèn LED khi nhấn nút số 1



Hình : Trạng thái khi nhấn nút số 1

Trạng thái đèn LED khi nhấn phím 2



Hình : Trạng thái ban đầu của 3 đèn LED

# : KẾT LUẬN

### Kết quả đạt được

Sau khi quá thực hiện, đề tài đã đạt được một số kết quả đáng khích lệ

* Tìm hiểu được GEF.
* Tìm hiểu được EMF.
* Tìm hiểu được lược đồ trạng thái.
* Tìm hiểu cấu trúc của Arduino.
* Tìm hiểu được cách kết nối thiết bị với Arduino.
* Tìm hiểu được cấu trúc chương trình chạy trên thiết bị Arduino.
* Xây dựng meta model sử dụng EMF
* Xây dựng giao diện cho người dùng (GUI) sử dụng GEF.
* Kết hợp EMF với GEF để tạo thành công cụ phát sinh mã nguồn WAGEN Tools.
* Xây dựng bộ cú pháp cho phép ánh xạ các mô tả trong lược đồ trạng thái với thiết bị trên bo mạch.
* Xây dựng bộ phát sinh mã nguồn cho ứng dụng chạy trên thiết bị wearable.
* Đã kiểm chứng ứng dụng điều khiển đèn led bằng keypad, ứng dụng quản lý đặt món.

### .Hạn chế của đồ án

* Chỉ phát sinh được chuyển các trạng thái cho mô tả ứng dụng đơn giản.
* Hạn chế về các thiết bị input, output.
* Lược đồ trạng thái chỉ hỗ trợ các xử lý, mô tả đơn giản.
* Chỉ phát sinh mã cho Arduino UNO R3.
* Mã nguồn phát sinh theo ngôn ngữ lập trình C.

### Hướng phát triển

* Thêm nhiều bo mạch Arduino khác vào hệ thống gợi ý người dùng chọn dùng loại Arduino nào khi tạo project.
* Thêm nhiều thiết bị input, output vào phần Mainboard đề có thể phát sinh mã điều khiển nhiều loại thiết bị.
* Xử lý ngữ nghĩa cho lược đồ trạng thái một cách hiệu quả cho ứng dụng.
* Thêm các chức năng cho giao diện đồ họa, giúp người dùng dễ sử dụng hơn.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| **[1]** | YAKINDU, "YAKINDU," [Online]. Available: https://www.itemis.com/en/yakindu/state-machine/. [Accessed 7 5 2018]. |
| **[2]** | "eclipse.org," [Online]. Available: http://www.eclipse.org/modeling/emf/. [Accessed 7 5 2018]. |
| **[3]** | "wiki.eclipse.org," [Online]. Available: https://wiki.eclipse.org/GEF\_Description#EditPartViewer.2C\_RootEditPart. [Accessed 7 5 2018]. |
| **[4]** | "marketplace," [Online]. Available: https://marketplace.eclipse.org/content/ant-visualization. [Accessed 7 5 2018]. |
| **[5]** | "marketplace," [Online]. Available: https://marketplace.eclipse.org/content/solunar-gmbh. [Accessed 7 5 2018]. |
| **[6]** | "https://eclipsesource.com," [Online]. Available: https://eclipsesource.com/blogs/tutorials/emf-tutorial/. [Accessed 7 5 2018]. |
| **[7]** | "http://www.vogella.com," [Online]. Available: http://www.vogella.com/tutorials/EclipseEMF/article.html. [Accessed 7 5 2018]. |
| **[8]** | "https://www.eclipse.org," [Online]. Available: https://www.eclipse.org/gef/. [Accessed 7 5 2018]. |
| **[9]** | "https://www.linuxtopia.org," [Online]. Available: https://www.linuxtopia.org/online\_books/eclipse\_documentation/eclipse\_gef\_draw2d\_plug-in/topic/org.eclipse.gef.doc.isv/guide/eclipse\_gef\_draw2d\_guide.html. [Accessed 7 5 2018]. |
| **[10]** | "https://view.officeapps.live.com," [Online]. Available: https://view.officeapps.live.com/op/view.aspx?src=https://www.fer.unizg.hr/\_download/repository/Gef.doc. [Accessed 7 5 2018]. |
| **[11]** | "https://wiki.eclipse.org," [Online]. Available: https://wiki.eclipse.org/GEF\_Description2. [Accessed 7 5 2018]. |
| **[12]** | "http://arduino.vn," [Online]. Available: http://arduino.vn/bai-viet/42-arduino-uno-r3-la-gi. [Accessed 7 5 2018]. |
| **[13]** | "https://vi.wikipedia.org," [Online]. Available: https://vi.wikipedia.org/wiki/M%C3%A1y\_tr%E1%BA%A1ng\_th%C3%A1i\_tr%E1%BB%ABu\_t%C6%B0%E1%BB%A3ng. [Accessed 7 5 2018]. |
| **[14]** | "https://help.eclipse.org," [Online]. Available: https://help.eclipse.org/mars/index.jsp?nav=%2F2\_0. [Accessed 7 5 2018]. |
| **[15]** | "http://arduino.vn," [Online]. Available: http://arduino.vn/bai-viet/77-bai-2-cach-lam-den-led-nhap-nhay-theo-yeu-cau. [Accessed 11 5 2018]. |
| **[16]** | "http://arduino.vn," [Online]. Available: http://arduino.vn/bai-viet/915-huong-dan-su-dung-module-ban-phim-4x4-voi-arduino. [Accessed 7 5 2018]. |
| **[17]** | Phuoc, Ton Long; Son, Le Lam; Anh, Pham Hoang;, "Towards a Domain Specific Framework for Wearable Applications in Internet of Things," Ho Chi Minh . |
| **[18]** | A. Apon, "Lecture for Cluster and Grid Computing," 2004. |
| **[19]** | Dave, Steinberg; Frank, Budinsky; Marcelo, Paternostro;, "EMF: Eclipse Modeling Framework, Second Edition," 2008. |
| **[20]** | "www.itemis.com," [Online]. Available: https://www.itemis.com/en/yakindu/state-machine/. [Accessed 7 5 2018]. |
| **[21]** | [Online]. Available: https://www.eclipse.org/articles/Article-GEF-editor/gef-schema-editor.html. [Accessed 7 5 2018]. |
| **[22]** | "www.vainolo.com," [Online]. Available: https://www.vainolo.com/tutorials/gef-tutorials/. [Accessed 7 5 2018]. |
| **[23]** | "www.eclipse.org," [Online]. Available: http://www.eclipse.org/articles/Article-GEF-diagram-editor/shape.html. [Accessed 7 5 2018]. |
| **[24]** | "https://www.programcreek.com," [Online]. Available: https://www.programcreek.com/java-api-examples/index.php?api=org.eclipse.draw2d.IFigure. [Accessed 7 5 2018]. |
| **[25]** | "https://help.eclipse.org," [Online]. Available: https://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.draw2d.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fdraw2d%2FIFigure.html. [Accessed 7 5 2018]. |
| **[26]** | "https://help.eclipse.org," [Online]. Available: https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.gef.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fgef%2Fui%2Fparts%2FGraphicalEditor.html. [Accessed 7 5 2018]. |
| **[27]** | "http://download.eclipse.org," [Online]. Available: http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details. [Accessed 7 5 2018]. |
| **[28]** | "http://www.philmann-dark," [Online]. Available: http://www.philmann-dark.de/EMFDocs/tutorial.html. [Accessed 7 5 2018]. |

# PHỤ LỤC

## Kế hoạch

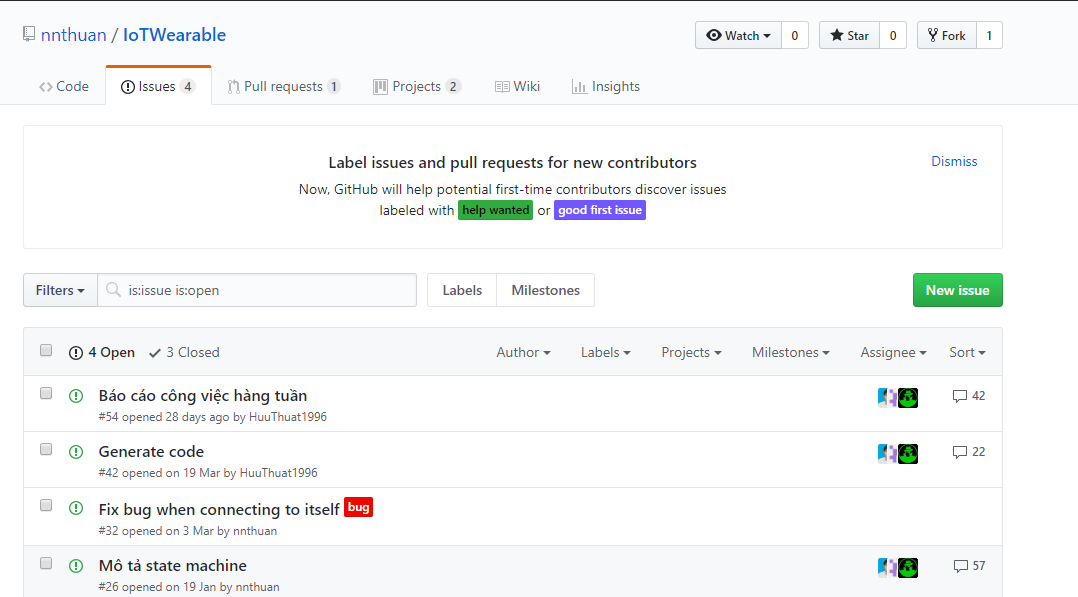
|  |  |  |
| --- | --- | --- |
| CÔNG VIỆC | THỜI GIAN | NGÀY BẮT ĐẦU |
| Tìm hiểu về EMF và GEF, Plug-in trong eclipse, tìm hiểu về Arduino và cách viết code đơn giản, một số thiết bị, linh kiện điện tử trong ứng dụng. | 7 | 12/1/2017 |
| Bắt đầu xây dụng công cụ IoT wearable giai đoạn thử nghiệm với việc thiết kế model sử dụng UML diagram và tạo dựa án trên github. | 1 | 12/9/2017 |
| Tạo plug-in model và plug-gef. | 2 | 12/10/2017 |
| Tạo các mô hình cho Switch, LCD, Buzzer sử dung EMF. | 1 | 12/10/2017 |
| Hiển thị model lên editor đồ họa sử dụng GEF. | 1 | 12/13/2017 |
| Tạo model cho statemachine và hiển thị đồ họa lên editor. | 2 | 12/14/2017 |
| Tạo kết nối (link) cho statemachine. | 1 | 12/15/2017 |
| Xây dựng bộ quy tắt, ngôn ngữ sử dụng statemachine. | 7 | 12/16/2017 |
| Xây dựng bộ phát sinh mã. | 21 | 12/16/2017 |
| Tạo bảng hướng dẫn. | 2 | 3/11/2018 |
| Kết thúc giải đoạn thử nghiệm bắt đầu với một buổi họp. | 1 | 3/24/2018 |
| Viết báo cáo giữa kỳ | 7 | 3/20/2018 |
| Chỉnh sửa và phát triển thêm model. |  | 3/20/2018 |
| Chỉnh sửa và phát triển thêm cho gef. |  | 3/20/2018 |
| Xây dựng bộ phát sinh mã. |  | 3/20/2018 |
| Phát sinh tài liệu hướng dẫn lắp đặt cho ứng dụng được thiết kết. |  | 3/20/2018 |
| Xây dựng các validation. |  | 3/20/2018 |
| Xây dựng các context menu. |  | 3/20/2018 |
| Viết báo cáo cuối kỳ | 7 | 5/1/2018 |

Bảng 0‑1: Bảng kế hoạch thực hiện đề tài

## Nhật ký

Nhóm dùng Github để quản lý source code, phân công công việc, tương tác với giáo viên hướng dẫn: <https://github.com/nnthuan/IoTWearable>.

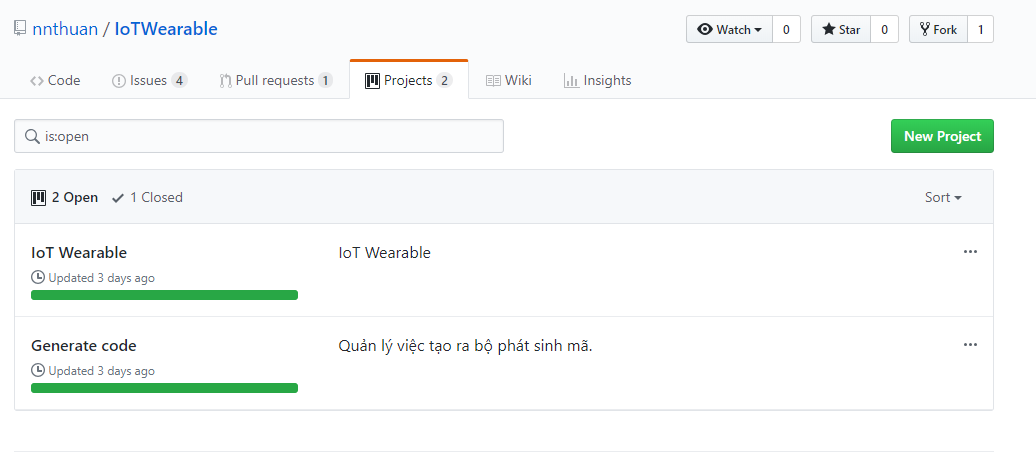
Nơi tương tác với giáo viên hướng dẫn: giao viên đưa ra yêu cầu, thảo luận các vấn đề của đề tài.



Hình : Danh sách các mục trao đổi với GVHD

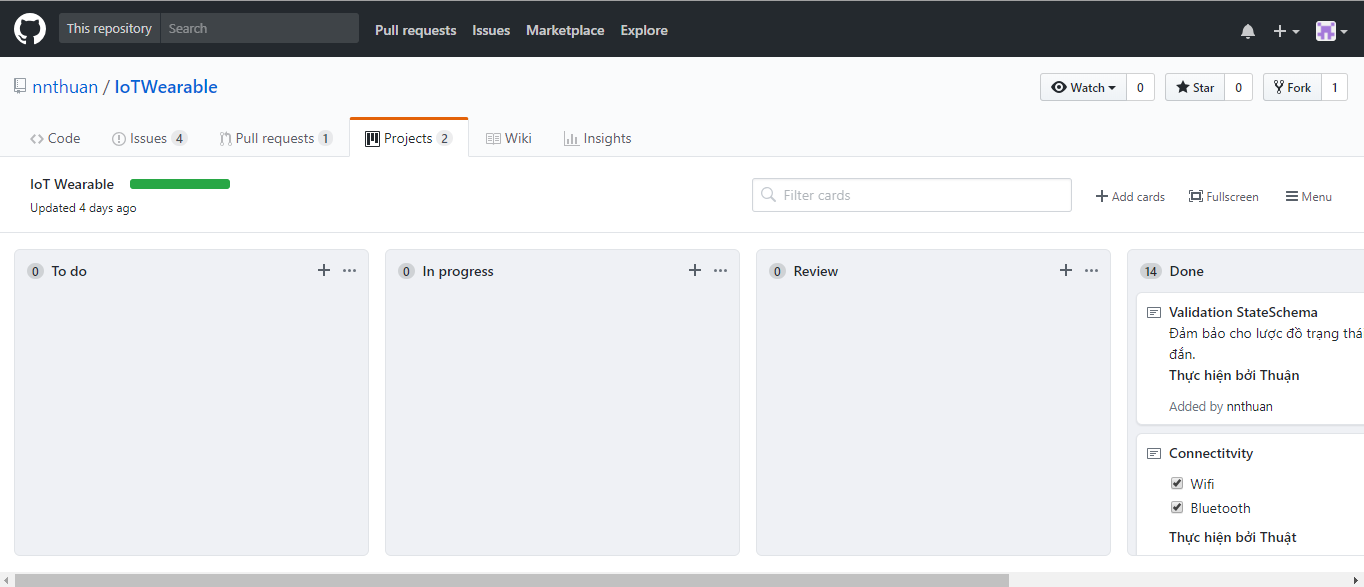
Phân công công việc theo cấu trục 4 cột:

1. To do: các công việc cần làm.
2. In progress: các công việc đang được làm.
3. Review: các công việc đã hoàn thành chờ các thành viên xem xét để quyết định hoàn thành hoặc tiếp tục làm.
4. Done: các công việc đã hoàn thành.



Hình : Danh mục các công việc

Quản lý công việc



Hình : Cách tổ chức quản lý công việc