**CHƯƠNG 2: PHẦN MỀM CƠ SỞ DỮ LIỆU CAN**

**2.1.           Giới thiệu chương.**

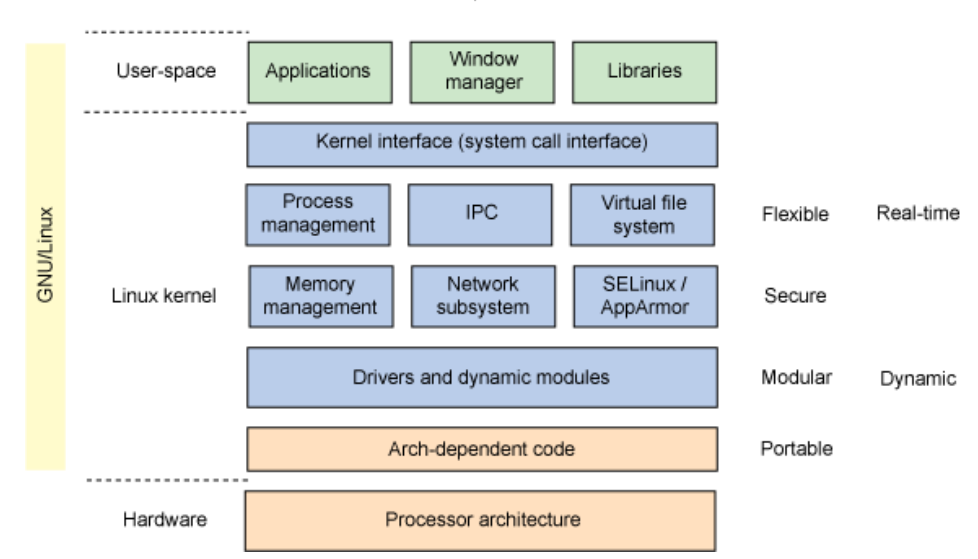
Chương này trình bày về những công cụ được sử dụng để tạo thành phần mềm cơ sở dữ liệu Can trong phần mềm mô phỏng hệ thống Can Bus.

**2.2.           Công cụ thực hiện**.

**2.2.1.     Môi trường hệ điều hành Linux.**

**2.2.1.1. Khái niệm:**

Nhìn bề ngoài, Linux là một hệ điều hành. Như thể hiện trong Hình 2.1, Linux gồm có một nhân kernel (mã cốt lõi quản lý các tài nguyên phần cứng và phần mềm) và một bộ sưu tập các ứng dụng của người dùng (chẳng hạn như các thư viện, các trình quản lý cửa sổ và các ứng dụng).



Hình 2.1. Kiến trúc của hệ điều hành Linux

Sơ đồ trên chỉ ra các thành phần quan trọng. Tầng cuối cùng chính là một tập hợp mã kiến trúc giúp Linux có thể hỗ trợ đa nền tảng phần cứng (ARM, PowerPC, Tilera TILE v.v...). Tất nhiên, chức năng này được đăng ký theo giấy phép GNU, tạo nên tính di động của Linux.

**2.2.1.2. Ưu điểm của của hệ điều hành.**

* **Bản quyền:**

Lợi thế của Linux chính là nền tảng mã nguồn mở và miễn phí.

* **Bảo mật**

Nếu như trên Windows bạn luôn phải chật vật đối mặt với ngày càng nhiều những con virus, mã độc,… thì bạn lại được an toàn khi sử dụng Linux, bởi vì đơn giản, tất cả bọn chúng đều không thể hoạt động được trên nền tảng này. Công việc của bạn chỉ là xóa khi thấy bọn chúng trong USB hay ổ cứng di động.

* **Linh hoạt**

Trên Linux, nếu bạn có nhiều hiểu biết về nó, bạn có thể dễ dàng chỉnh sửa theo ý mình. Hơn nữa, Linux còn mang lại sự tương thích với rất nhiều môi trường khác nhau và đây là một môi trường lý tưởng cho các lập trình viên cũng như các nhà phát triển.

* **Hoạt động mượt mà trên các máy tính có cấu hình yếu**

Hệ điều hành này hoạt động mượt mà và cực kỳ ổn định trên các máy tính có cấu hình thấp và vẫn được nâng cấp, hỗ trợ thường xuyên từ cộng đồng lập trình Linux.

**2.2.2.     Giới thiệu chung về Framework.**

* Framework là một thư viện các lớp đã được xây dựng hoàn chỉnh , bộ khung để phát triển các Phần mềm ứng dụng. có thể ví framework như tập các “Vật liệu” ở từng lĩnh vực cho người lập trình viên, thay vì họ phải mất nhiều thời gian để tự thiết kế trước khi dùng. Do vậy, người lập trình viên chỉ cần tìm hiểu và khai thác các vật liệu này rồi thực hiện ( tức lập trình ) để gắn kết chúng lại với nhau, tạo ra sản phẩm.
* Mục tiêu của các nhà phát triển nên Qt chính là tạo ra một nền tảng có khả năng thiết kế những phần mềm có thể chạy trên nhiều nền tảng phần mềm lẫn phần cứng khác nhau mà không phải thay đổi nhiều về code.

**2.2.3.     Các ưu điểm của QT Framework**

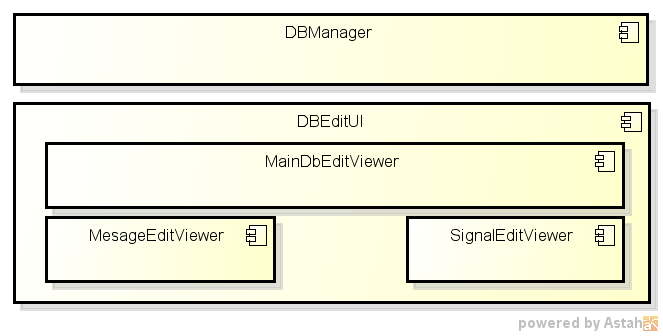
* Qt hổ trợ các nền tảng sau: Windows,Linux, OS X, iOS, Android, WinRT, …
* Qt hổ trợ ngôn ngữ C/C++/Java/Python/…
* Qt hổ trợ lập trình giao diện chuẩn theo mô hình model/view và hổ trợ các lớp lập trình giao diện động trong Qt Quick, giúp tạo giao diện chuyên nghiệp và sinh động hơn.

**2.3.           Phần mềm cơ sở dữ liệu CAN.**

**2.3.1.     Tổng quan phần mềm**

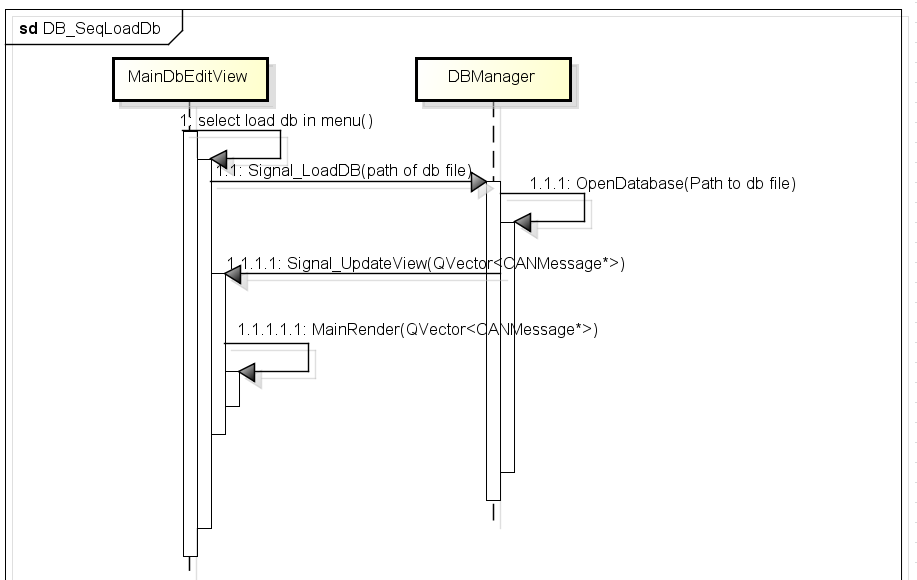
Phần mềm bao gồm 2 phần chính:

* Cơ sở dữ liệu
* Giao diện
* Giao diện chính chỉnh sửa database
* Giao diện chỉnh sửa message
* Giao diện chỉnh sửa Signal

Phần mềm cho phép tải dữ liệu từ file .dbc lên giao diện chính với các thông số dữ liệu của Message và Signal. Tại từng message và signal có thể show được dữ liệu và edit toàn bộ các thông số tương ứng.

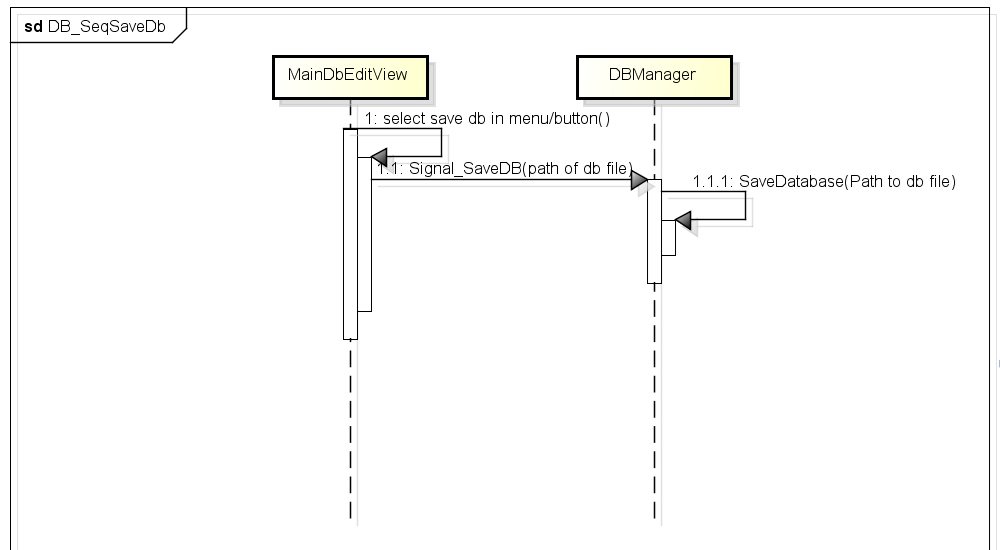
Hình 2.2. Database Editor

**2.3.2.     Tiến trình thực hiện**

* Load database

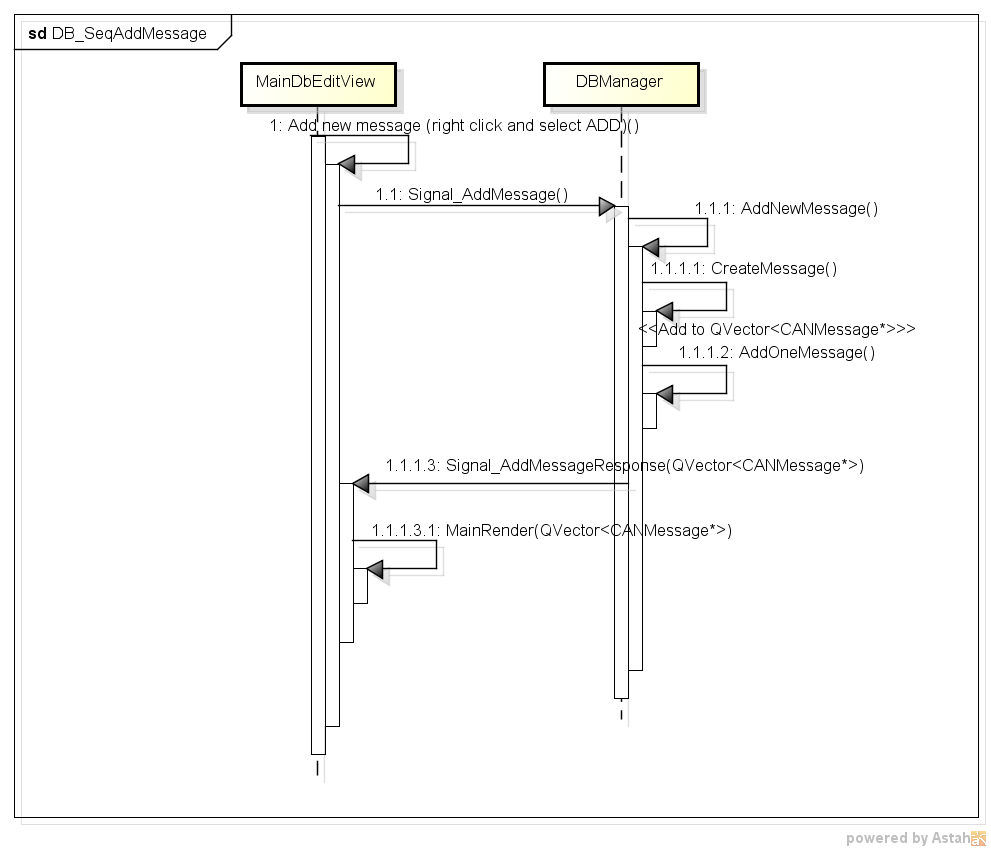
Hình 2.3. Sequence diagram LoadDB

* Save database



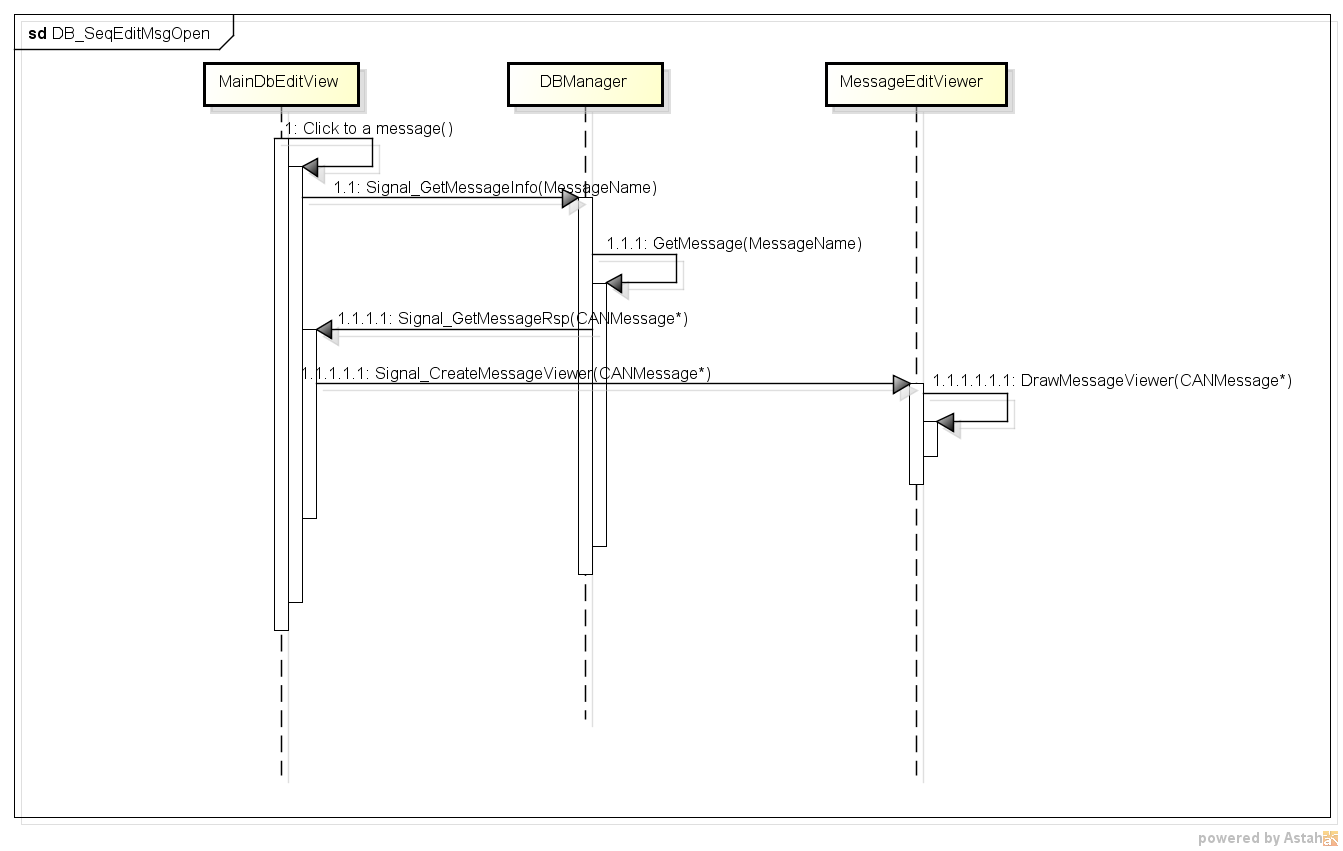
Hình 2.4. Sequence diagram SaveDB

* Add Message



Hình 2.5. Sequence diagram Thêm Message

* Open edit Message



Hình 2.6. Sequence diagram Mở dialog chỉnh sửa message

**2.3.5.     Mô hình Model/View trong QT Framework**

* Kiến trúc MVC và kiến trúc model/view trong QT:
* Mô hình MVC (Model-View-Controller) là một kiến trúc phần mền, nó giúp cho các nhà phát triển phần mền tách các ứng dụng của họ ra thành 3 phần có nhiệm vụ riêng biệt và độc lập với các thành phần khác, trong đó:
* Model: chứa các phương thức xử lí và truy xuất dữ liệu.
* View: Đảm bảo việc hiển thị thông tin và tương tác với người dùng.
* Controller: là trung gian giữa Model và View, nhận các yêu cầu từ tương tác người dùng từ view và tác động làm thay đổi dữ liệu trong model.
* QT framework sử dụng mô hình model/view thay cho mô hình MVC, trong đó lớp Controller sẽ được tích hợp trong View thông qua Delegate.



Hình 2.7. Cấu trúc model/view trong QT

* **Lí do lựa chọn mô hình model/view:** mô hình này chia phần mền thành các module riêng biệt, nên sẽ dễ dàng hơn cho việc bảo trì, nâng cấp, phát triển sau này.
* Hoạt động của model/view:
* Ban đầu model sẽ sử dụng phương thức setModelData () của mình để lấy dữ liệu từ tập dữ liệu gốc (dataset ), và View sẽ tự cập nhật dữ liệu được lấy từ model thông qua phương thức setModel().
* Mỗi tương tác chỉnh sửa của người dùng trên giao diện sẽ gọi một đối tượng của 1 lớp Delegate tương ứng, thông qua delegate sẽ làm thay đổi dữ liệu tương ứng trong model, khi đó model sẽ cập nhật hiển thị mới trên View cũng như phát đi một signal để làm thay đổi dữ liệu ở dữ liệu gốc, signal này được kết nối với một slot tương ứng ở lớp quản lí tập dữ liệu gốc, slot này sẽ làm thay đổi dữ liệu trong dataset.

**2.3.6.     Cơ chế Signal và Slot trong QT Framework**

* Signal và Slot trong QT:
* Signal: khi một sự kiện nào đó xảy ra, một signal sẽ được phát đi, thực ra nó chỉ là một phương thức của một lớp nhưng không có phần thân hàm {}. Các lớp Widget có sẵn trong Qt có rất nhiều signal được định nghĩa sẵn, và chúng ta cũng có thể viết các signal riêng cho các lớp của tự định nghĩa. Signal không có kiểu trả về, kiểu trả về của signal luôn luôn là void.
* Slot:  là một phương thức bình thường của một lớp, các phương thức này sẽ được gọi khi có một signal nào đó được phát đi. Cũng giống như signal, các lớp Widget trong Qt cũng có sẵn rất nhiều slot và chúng ta cũng có thể viết slot cho lớp của riêng chúng ta.
* Connect: Signal và slot được kết nối qua từng đối tượng thông qua phương thức connect.

Connect (Đối tượng 1, SIGNAL (signal thuộc đối tượng 1), Đối tượng 2, SLOT (slot thuộc đối tượng 2));

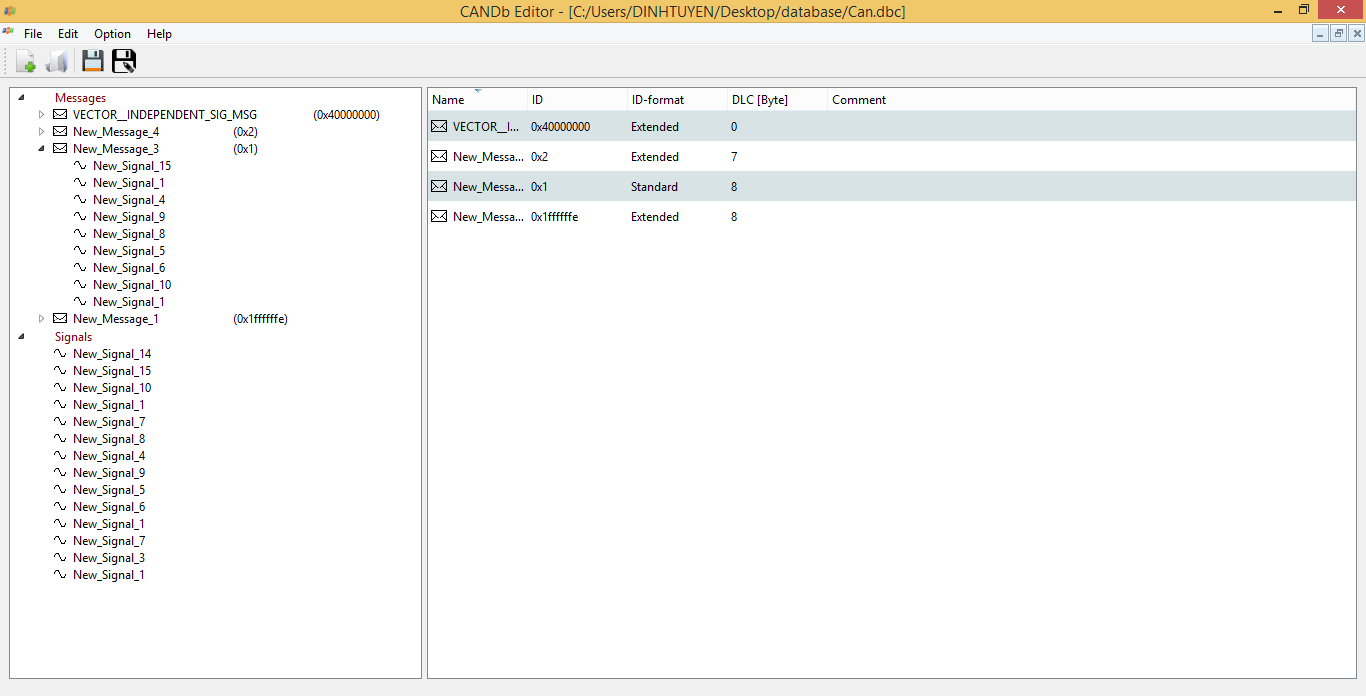
Sau khi kết nối sẽ có một đối tượng phát ra signal và một đối tượng nhận signal đó và thực hiện một phương thwucs tương ứng, lúc này phương thức đó sẽ là một slot. Kết nối signal và slot là một cách tiện lợi để người dùng gửi đi các signal làm thay đổi dữ liệu tương ứng trên dữ liệu gốc cũng như trên model thay cho delegate.

* Để có thể kết nối signal và slot thì hoặc là phương thức slot của đối tượng nhận signal phải có thành phần tham số giống với signal hoặc là slot phải không có tham số.

**2.3.7.     Thiết kế phần mền cơ sở dữ liệu CAN.**

**2.3.7.1     Thiết kế giao diện.**

* Phần mềm sẽ gồm 2 giao diện chính :
* Tại màn hình chính:



Hình 2.8. Giao diện chính phần mền cơ sở dữ liệu CAN bus

Giao diện chính của CAN database gồm 2 phần chính:

* Một danh sách dạng cây quản lí tên của các message và signal:

Trong phần này, ta sẽ sử dụng mô hình model/view gồm các thành phần:

Model: kế thừa lớp QAbstractItemModel để tạo QMainDBTreeViewModel, trong đó chứa các hàm xử lí dữ liệu từ datasource.

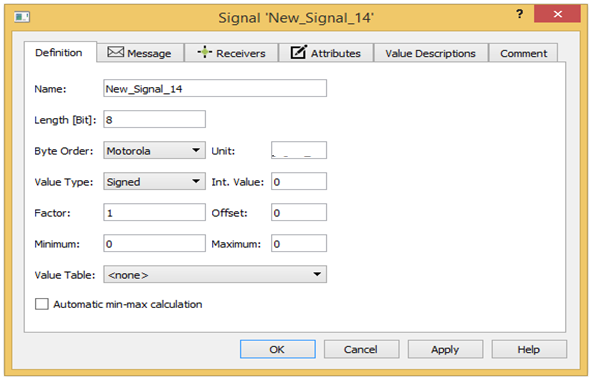
View: sử dụng QTreeView để hiển thị dữ liệu ở trong model

* Phần hiển thị thông tin chi tiết gồm:
* Các bảng chứa thông tin chi tiết của các message(hiển thị khi nhấn chuột vào message ở QTreeView)
* Các bảng chứa thông tin chi tiết của các signal(hiển thị khi nhấn chuột vào signal ở QTreeView)
* Các bảng chứa thông tin chi tiết của các signal trong message (hiển thị khi nhấn chuột vào từng message , signal hoặc signal trong message ở QTreeView)

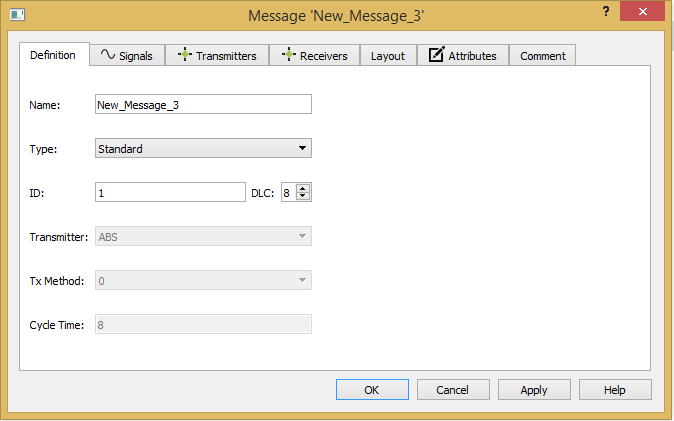
Cả 3 bảng trên ta đều sử dụng lớp QTableView phần View, Model của các bảng sẽ được thực hiện bằng cách kế thừa từ lớp QAbstractItemModel để tạo ra các TableModel tương ứng với 3 view ở trên. Trong mỗi bảng sẽ có các lớp Delegate tương ứng được kế thừa từ lớp QAbstractItemDelegate, khi người dùng tương tác với View, View sẽ thông qua delegate để chỉnh sửa được dữ liệu.

**2.3.7.2     Thiết kế phần xử lí dữ liệu CAN.**

* Màn hình chỉnh sửa : message editor và signal editor



Hình 2.9. Giao diện signal editor



Hình 2.10. Giao diện message editor



Hình 2.11. Giao diện message signal editor

Giao diện chỉnh sửa sẽ có nhiều trang(tab), mỗi trang sẽ chứa nhiều thông số khác nhau , lúc nhấn Ok hoặc Apply thì thông tin tất cả các thông tin trong các trang chỉnh sửa sẽ được lưu lại .

* Đối với giao diện này ta sẽ sử dụng lớp QTabWidget trong QT, hổ trợ các phương thức addTab (QWidget \* page, const QString & label) để thêm các trang mới.
* Đối với nội dung trong từng trang ta sẽ sử dụng các lớp QLabel để đặt tên của thuộc tính, và sử dụng QComboBox, QLineEdit để hiển thị giá trị tương ứng với giá trị là các bảng lựa chọn hay text
* Để có được định dạng và sắp xếp hợp lí, ta sẽ sử dụng QLayout và các lớp con QHBoxLayout, QVBoxLayout để layout cho giao diện.

Khi người dùng thao tác chỉnh sửa xong và nhấn một trong các nút Ok, Apply, Cancel hay Help thì các signal tương ứng sẽ được tạo ra và các lớp Model và DBMainManager sẽ nhận những signal này thông qua các slot, từ đó dữ liệu sẽ được thay đổi tùy thuộc vào việc xử lí trong các slot.

2.3.10. Kết luận chương