Modelagem de um ECG com o framework CHESS

Este documento contém de forma detalhada o processo de modelagem e análise dos resultados de confiabilidade e disponibilidade de um ECG para um ambiente de UTI cardíaca.

Este documento está organizado primeiramente na modelagem do ECG, contendo as visões a nível de sistema e também a nível de *software*, e após isso como foi aplicada as análises de confiabilidade e disponibilidade nos modelos já desenvolvidos do ECG. Também neste mesmo documento, existe a estrutura de como a modelagem esta organizada arquiteturalmente dentro do CHESS.

System View

a descrever...

Software View

A visão de *software* é responsável por proporciar que o modelador realize abstrações do que se pretende modelar a nível de *software*. Para o ECG foram realizadas 5 abstrações para modelagem, que são:

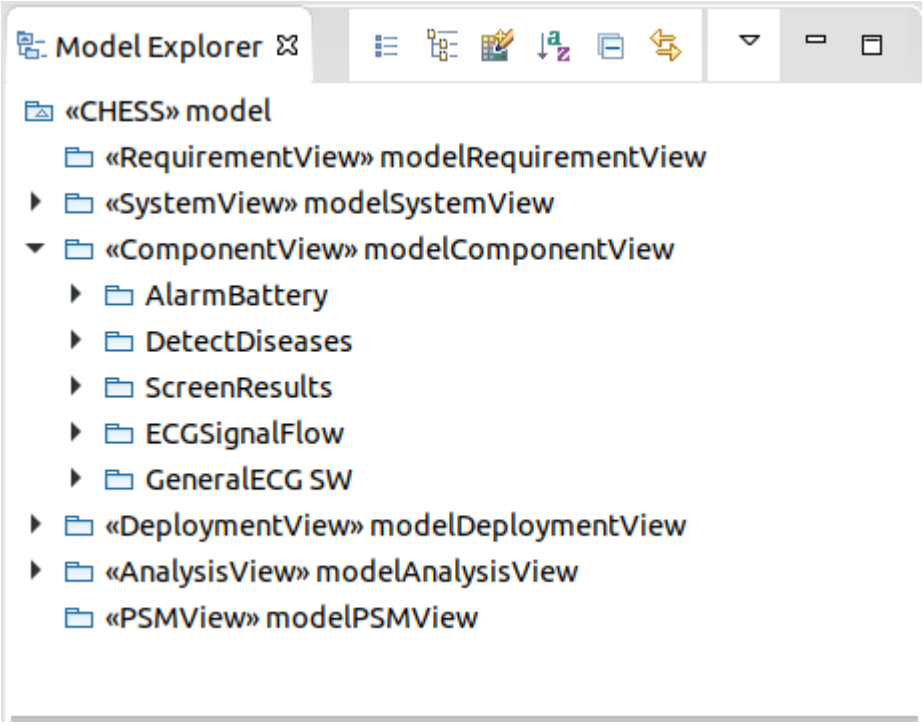
1. Detecção de doenças;
2. Captura de sinal com os eletrodos;
3. Alarme de bateria e detecção de carga;
4. Fluxo do sinal capturado dentro do ECG;
5. Tela de resultados que o ECG coletou de um ou mais pacientes.

A partir dessas modelagens, foi desenvolvida uma modelagem de nível superior conectando todas essas modelagens, e definindo um fluxo para as modelagens definidas de nível inferior.

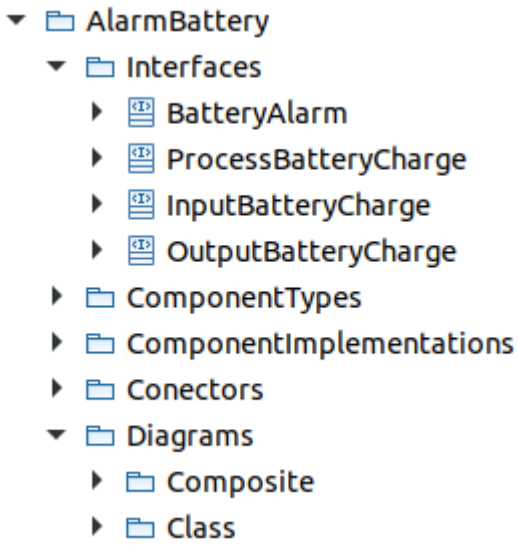
Estrutura da modelagem

Esta modelagem necessita possuir uma organização arquitetural, assim facilita a manutenabilidade e o entedimento de terceiros ao ver a modelagem e consigar localizar o que for necessário. A nível de software, a modelagem do ECG foi organizada prioritariamente e separada em packages representando

as principais modelagens. Dentro de cada package, irão ser encontradas os components, interfaces, diagramas e etc sobre cada modelagem.

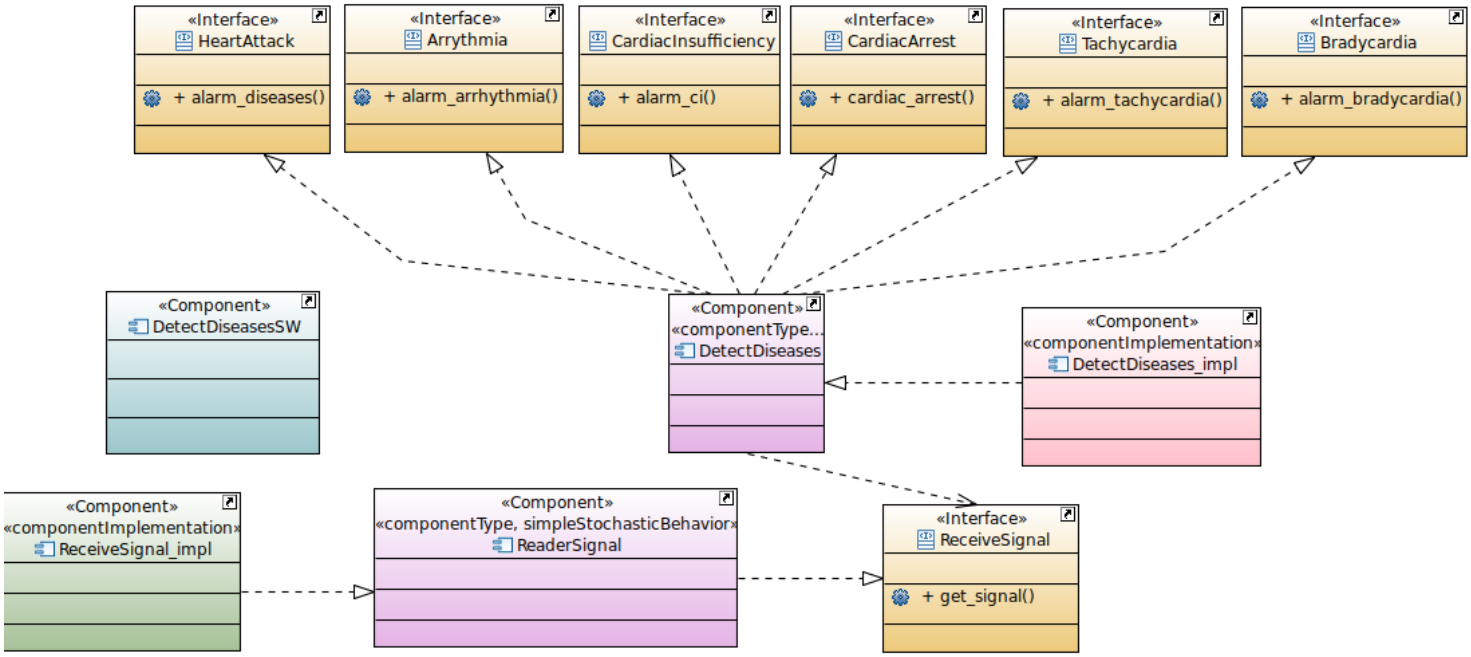


Dentro de cada modelagem, todos possuem o mesmo padrão. Ao explorar uma modelagem específica como por exemplo a modelagem de AlarmBattery, será possível perceber que existem pacotes que são responsáveis por armazenar as Interfaces, ComponentTypes, ComponentImplementations, Conectores e Diagramas, e dentro de diagramas separados em diagramas de compsição e diagramas de classes, como mostra a figura a seguir.

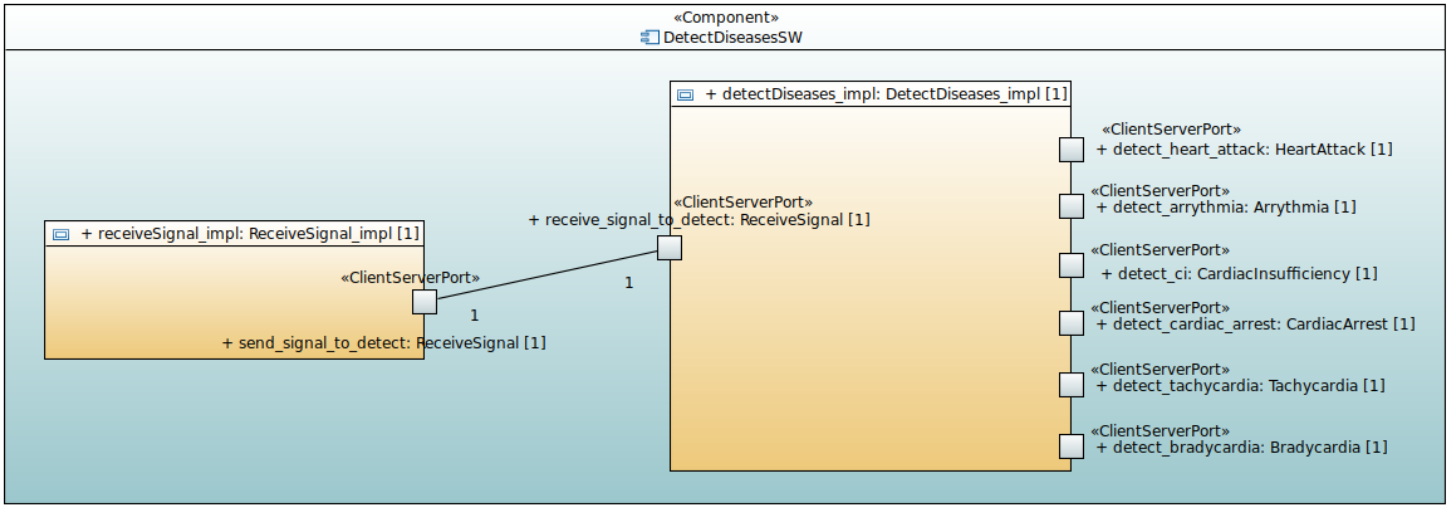


Detecção de doenças

Para a modelagem de detecção de doenças a nível de software, foi necessário primeiro criar um diagrama de classes para poder definir as *interfaces*, *components*, *componentsimplementations* e também as relações. As interfaces definidas representam a recepção do sinal para o software que irá detectar as doenças cardíacas, e também as interfaces representam quais doenças cardíacas esse software pode detectar. Assim existe um component para a detecção de doenças e assim uma implementação para ele, e também um component para receber o sinal que também possui uma implementação. Isso é possível observar na imagem.



Após definir o diagrama de classes para a detecção de doenças na visão de *software*, é necessário criar um diagrama de composição. Para isso é necessário definir no diagrama de classes um *component* que representa de um nível superior essa modelagem. Esse component definido foi nomeado de *DetectDiseasesSW*. No diagrama de composição este *component* possui os *component implementations*, que estão relacionados com os seus respectivos *components* e interfaces. Os *component implementations* definidos no diagrama de classes são adicionados como *part* dentro do diagrama de composição, assim o mesmo *component implementation* definido no diagrama de classe, é alocado no diagrama de composição possuindo uma referência ao diagrama de classes onde foi definido originalmente.



A modelagem para detecção de doenças possui dois *components implementations*, *ReceiveSignal_impl* e *DetectDiseases_impl*. Esses *components implementations* estão diretamente ligados com os mesmos *component implementations* definidos no diagrama de classes, pois no diagrama de composição eles são *parts*, logo no diagrama de composição eles também herdam os relacionamentos com as interfaces e *components* do diagrama de classes.

O *component implementation ReceiveSignal_impl* possui uma porta de saída com a interface *ReceiveSignal* que ele está diretamente ligado, que é a *send_signal_to_detect* . Esta porta de saída tem conexão direta com uma porta de entrada do *component implementation DetectDiseases_impl*, que também é do tipo *ReceiveSignal*. O *component implementation DetectDiseases_impl* ao receber em uma porta de entrada o *ReceiveSignal*, é possível realizar a detecção de doenças cardíacas, e de acordo com a doença cardíaca detectada, cada uma pode ir para a sua porta de saída específica. Cada porta de saída do *DetectDiseases_impl* representa uma doença cardíaca detectada.

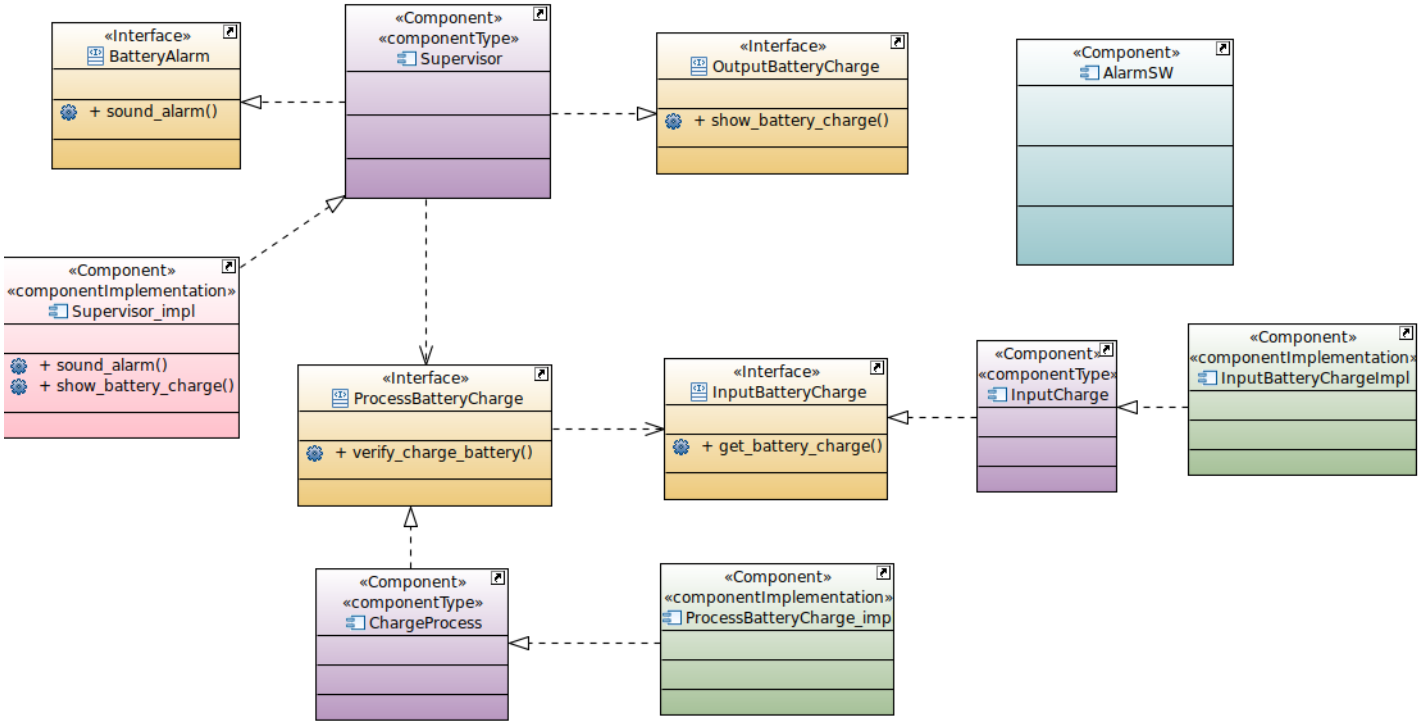
Eletrodos e captura de sinal

Modelagem em desenvolvimento...

Alarme da dateria e detecção de carga

Para a utilização do ECG, inicialmente o mesmo precisa verificar se a bateria possui uma carga suficiente para que o ECG inicie a captura do sinal, se a bateria estiver com uma carga abaixo do aceitável, o ECG não deve iniciar o procedimento de captura de sinal e deve soar um alarme informando que a bateria possui um baixo nível de carga. Se a bateria possuir uma carga aceitável para iniciar o procedimento, o ECG deve informar a carga da bateria e então iniciar o procedimento de captura de sinal normalmente.

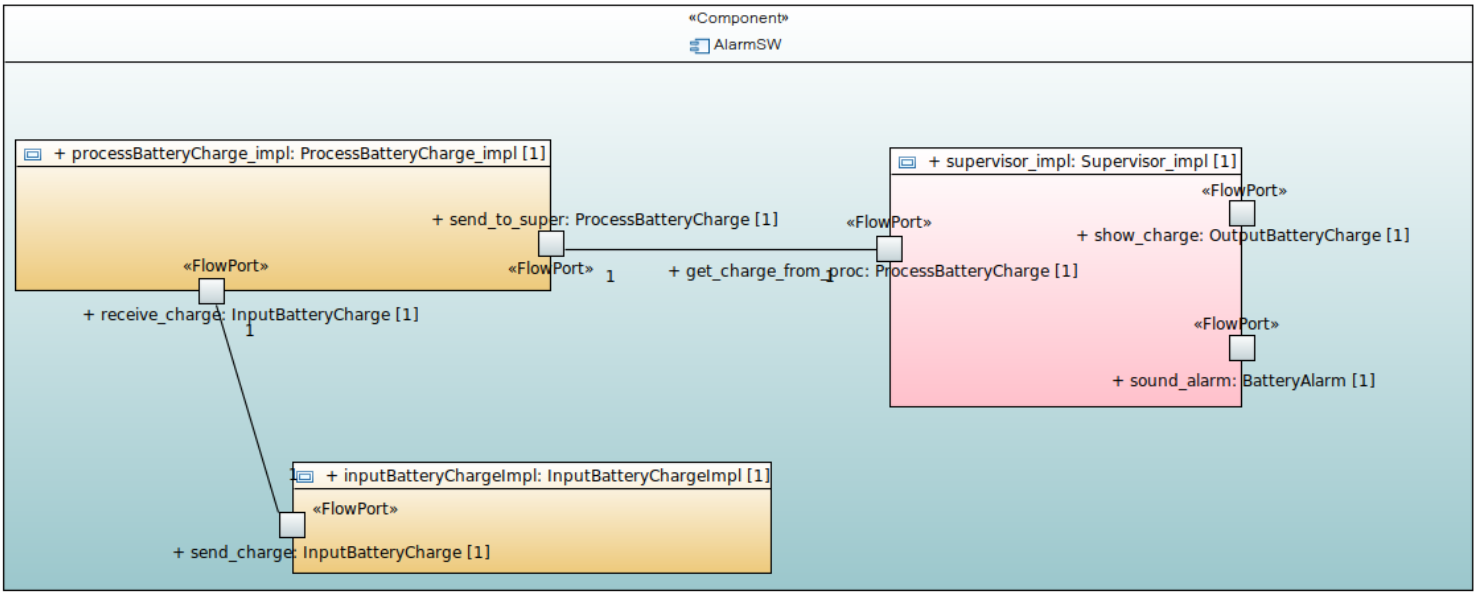
A detecção de carga de bateria ou alarme de bateria foi modelada da forma que, é necessário receber uma entrada com um valor de carga da bateria, processar esse valor de carga e então um supervisor fica responsável em identificar se a carga da bateria recebida e analisada é abaixo do normal para que o ECG não incie o procedimento de captura de sinal do paciente e emita um alarme informando que a bateria está com pouca carga, ou apenas informe que a bateria tem uma carga suficiente para que o ECG inicie o procedimento e então realiza a captura de sinal do paciente.



Foram definidas 4 interfaces para esta modelagem:

1. InputBatteryCharge
2. ProccessBatteryCharge
3. OutputBatteryCharge
4. BatteryAlarm

A interface *InputBatteryCharge* possui o seu *ComponentType* sua Implementação associadas diretamente, esse é o primeiro passo para que este fluxo seja executado. A interface *ProccessBatteryCharge* através do seu component e implementação recebem o valor da carga da bateria e realizam a verificação dessa carga para então informar ao supervisor que está relacionado as duas interfaces *OutputBatteryCharge* e *BatteryAlarm*, através da implementação será informado se a bateria possui carga suficiente para realizar o exame ou se é necessario emitir um alarme. O component *AlarmSW* é responsável por definir um diagrama de composição e então demonstrar a comunicação entre as implementações e as suas portas.

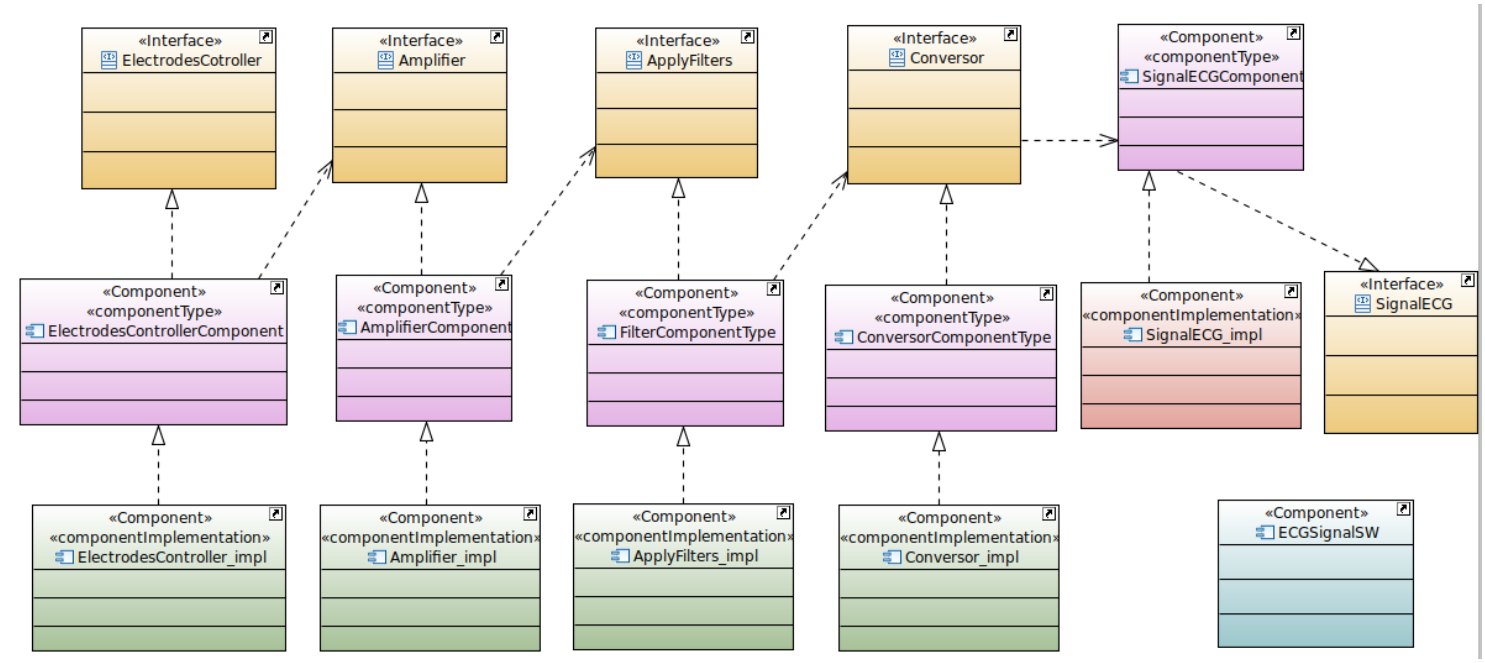


O diagrama de composição do *AlarmSW* inicia-se através do *inputBatteryChargeImpl* recebendo um valor de carga da bateria e enviando esse valor para a porta de saída nomeada de *send_charge* do tipo *InputBatteryCharge*, esta porta de saída possui uma comunicação direta com a porta *receive_charge* do mesmo tipo no *processBatteryCharge_impl*. O *processBatteryCharge_impl* ao receber o valor através da sua porta de entrada, pode aplicar suas operações nesse valor recebido e então encaminhar para a sua porta de saída *send_to_super* do tipo *ProcessBatteryCharge* e que possui comunicação direta com o *Supervisor_impl* através da porta *get_charge_from_proc*. Dentro do *Supervisor_imp*, ele irá decidir para onde encaminhar a informação, se será um alarme de pouca bateria emitido através da porta *sound_alarm* do tipo *BatteryAlarm* ou uma informação para a porta *show_charge* do tipo *OutputBatteryCharge*.

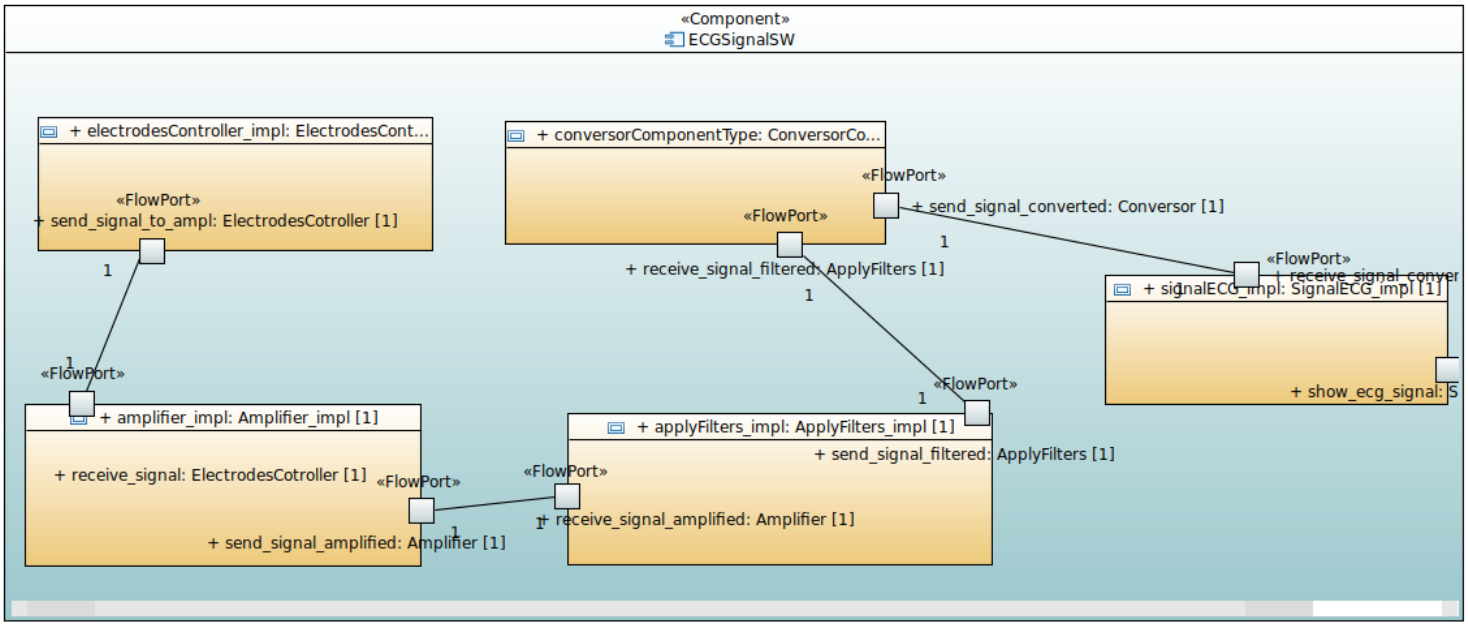
Fluxo do sinal capturado

Assim como nas outras modelagens a nível de *software*, foi necessário definir um diagrama de classes e um diagrama de composição. Foi necessário definir 5 interfaces, e assim seus respectivos *components* e *components implementations*. As interfaces definidas foram as seguintes:

1. ElectrodesController
2. Amplifier
3. ApplyFilters
4. Conversor
5. SignalECG



Todas essas interfaces possuem seus components e components implementations. A interface que gerencia todas as outras e que tem um nível hierárquico é a *SignalECG*, ela é a de nível mais alto nessa abstração, pois quando sua implementação é executada, ela irá precisar da implementação do conversor que por si só depdende da implementação do *ApplyFilter*, que depende da implementação do *Amplifier* e que por fim depende da implementação do *ElectrodesController*, e então o fluxo pode ser bem executado. Para representar melhor esse fluxo de depedências, foi definido um component nomeado de *ECGSignalSW*. Nesse component foi criado um diagrama de composição.

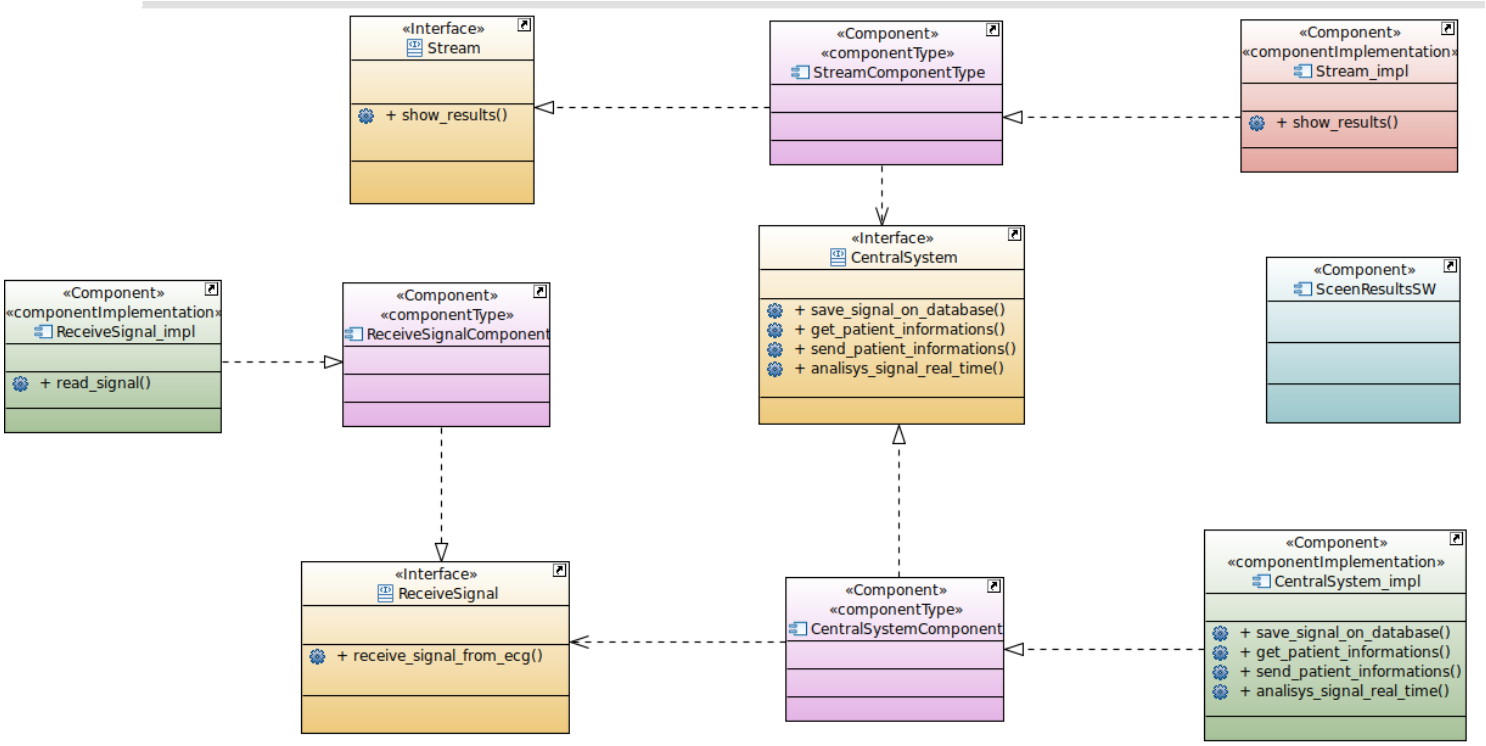


No diagrama de composição *ECGSignalSW* foram adicionados os *components implementations* como *parts* possuindo uma referência direta com o que foi definido originalmente no diagrama de classes. Assim como a ordem de dependência explicada no diagrama anterior, no diagrama de composição isso é mais legível e mais fácil de compreender. O fluxo se inicia através do *electrodesController_impl* que

possui uma porta de saída nomeada de *seng_signal_to_ampl* do tipo *ElectrodesController*, essa porta se comunica diratamente com a porta de entrada *receive_signal* do tipo *ElectrodesController* do *Amplifier_impl*. Dentro do *Amplifier_impl*, ele realiza as operações necessárias e então envia o sinal para uma porta de saída nomeada de *send_signal_amplified* que se comunica com a porta de entrada *receive_signal_amplified* do *ApplyFilters_impl*, possuindo o tipo *Amplifier*. O *ApplyFilters_impl* realiza algumas operações dentro do *component* e utiliza a porta de saída *send_signal_filtered* do tipo *ApplyFilters* para a porta de entrada *receive_signal_filtered* possuindo o tipo *ApplyFilters* no *Conversor_impl*. O *Conversor_impl* realiza a conversão do sinal e então permite que o sinal utilize sua porta de saída *send_signal_converted* do tipo *Conversor* para se comunicar com a porta de entrada *receive_signal_converted* do tipo *Conversor* do *SignalECG_impl*. O *SignalECG_impl* por fim possui uma porta de saída nomeada de *show_ecg_signal* do tipo *SignalECG* que representa o resultado dos sinais capturados por um ECG.

Tela de resultados

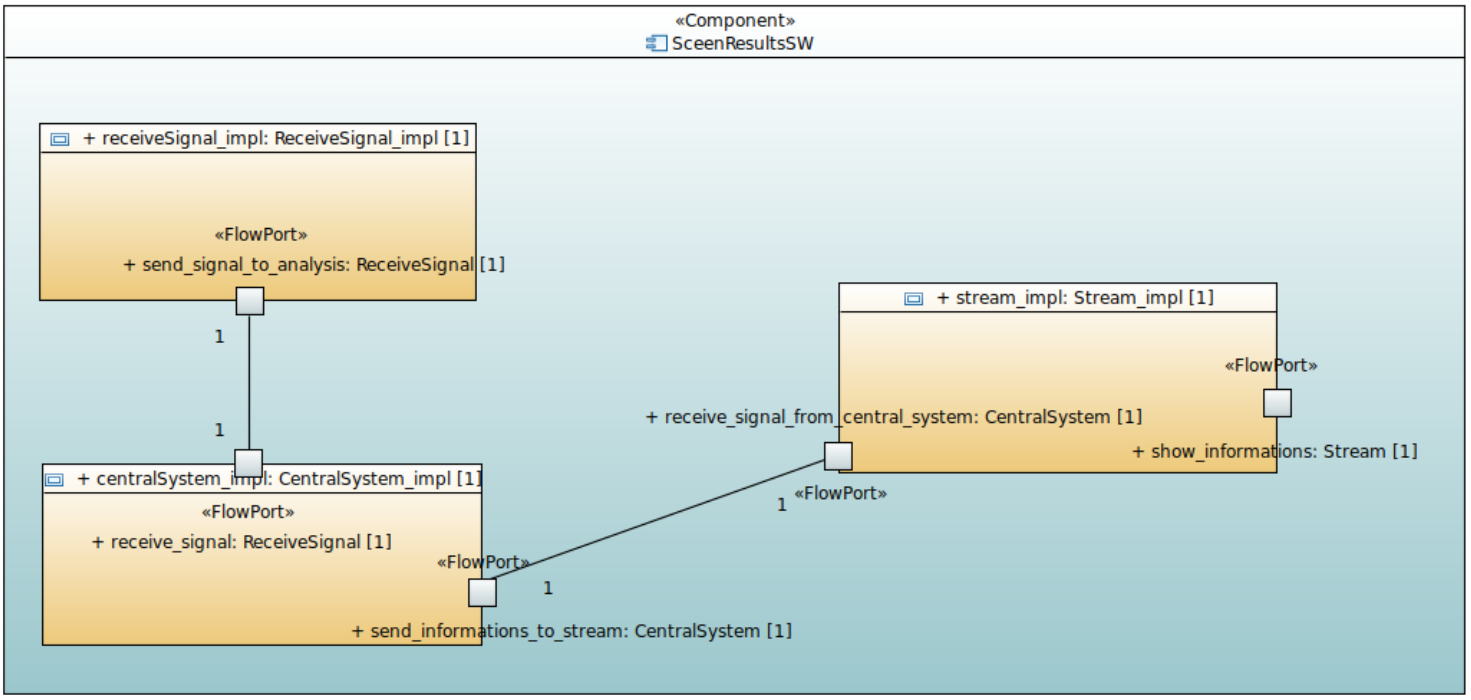
Em um ambiente de UTI cardíaca, os pacientes devem ser monitorados. Para facilitiar esse processo de monitoramento, os leitos possuem canais para a utilização do ECG, e o ECG coleta as informações cardíacas do paciente e envia para um servidor, onde os profissionais responsáveis pelo monitoramento como, por exemplo, enfermeiros e médicos podem monitorar, analisar e tomar decisão. Este servidor deve possuir uma tela de resultados com informações dos pacientes, salvar informações do paciente e recupera-las, analisar o sinal em tempo real, e ser capaz de emitir alarmes caso alguma anomalia urgente seja detectada naquele momento.



Para a tela de resultados, no diagrama de classes foram definidas 3 interfaces:

1. ReceiveSignal
2. CentralSystem
3. Stream

Após o ECG realizar todo o processo de captura de sinal, a interface *ReceiveSignal* é capaz de receber esse sinal e através da sua implementação poder ler e receber o sinal. A interface *CentralSystem* representa o servidor de fato, e é responsável por salvar o sinal em um banco de dados, obter informações do paciente, enviar informações do paciente e realizar análises em tempo real. Com o *CentralSystem* possuindo controle disso, a interface *Stream* possui relação de dependência com a *CentralSystem* para assim poder mostrar os resultados na tela de resultados. Um component *ScreenResultsSW* foi definido para ser responsável por representar todo esse fluxo através de um diagrama de composição.



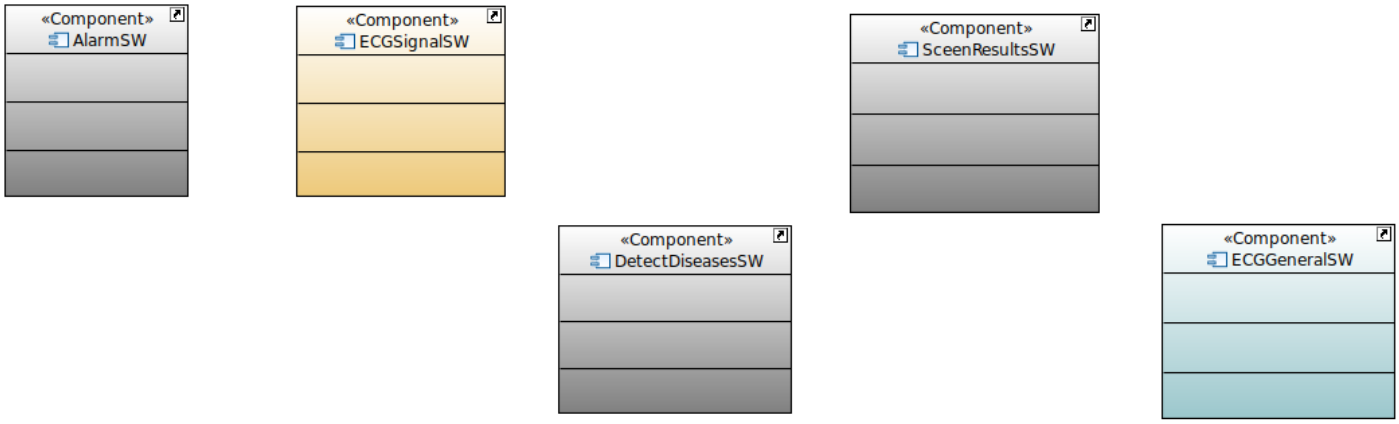
O diagrama de composição inicia seu processo através do *receiveSignal_impl*, onde possui uma porta de saída *send_signal_to_analysis* do tipo *ReceiveSignal* em comunicação direta com a porta *receive_signal* no *centralSystem_impl*. A partir do *CentralSystem_impl*, existe o envio das informações para a tela de resultados final, esse envio é representado através da comunicação entre as portas *send_informations_to_stream* no *CentralSystem_impl* para a porta *receive_signal_from_central_system* em *Stream_impl*, e então essa implementação é responsável por exibir as informações do paciente.

Visão geral do ECGSW

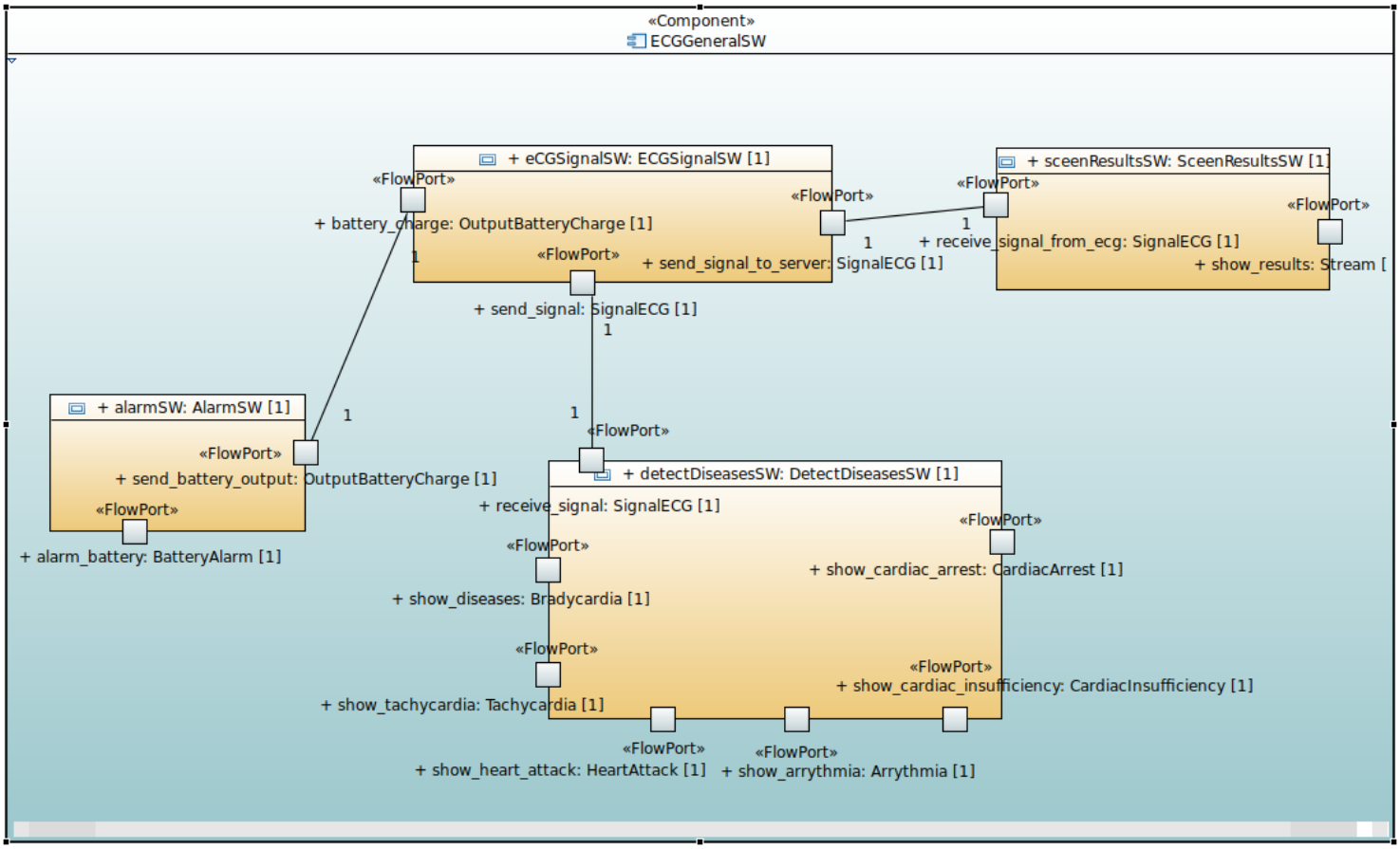
Esta modelagem é a modelagem de software de maior nível superior. Todas as outras modelagens já desenvolvidas vão ser conectadas através desta modelagem. Para isso, foi necessário referenciar

todos os components de nível superior de cada modelagem, ou seja os components resposáveis pelos diagramas de composições.

1. AlarmSW
2. ECGSignalSW
3. DetectDiseasesSW
4. ScreenResultsSW



Após adicionar os components de maior nível superior de cada modelagem, e que representam seus diagramas de composição, foi preciso também criar um component responsável para a modelagem de visão geral do ECG, e através dela poder realizar a conexão das outras modelagens e o diagrama de composição.



O diagrama de composição da visão geral do ECG representa as saídas que cada modelagem apresenta e possui a função de conectar uma modelagem com a outra. O inicio do fluxo inicia através da modelagem do *AlarmSW*, onde as saídas são um alarme caso a bateria seja baixa, ou uma informação com o valor de carga da bateria que logo é passada para o fluxo de sinal do ECG. A saída do fluxo de sinal do ECG podem ter dois caminhos, o primeiro é, caso alguma anomalia seja detectada, essa informação é enviada para a modelagem que é responsável por tratar doenças cardíacas e assim emitir um alarme para a doença cardíaca detectada, caso não o *ECGSignalSW* pode transmitir o sinal para a modelagem da tela de resultados, que então pode exibir os resultados em uma tela para os profissionais responsáveis por monitorar e analisar.