SOFTWARE DESIGN DETAILS

3.1 UML Diagrams for Utility Classes

The following diagrams represent the classes and methods within those classes that when called, execute tasks that will allow the Pacemaker to function. The diagrams also give insight into the permissions needed to access particular methods and variable values.

class_name
**variable: type
**method: type

main()
patientFirstName: private string patientLastName: private string patientAge: private uint18_t doctorNotes: private string

Pacemaker()
deviceID: private const string deviceImplantDate: private const string leadImplantDate: private const string replaceBattVoltage: protected const float batteryVoltage: protected float cardiacEvents: protected [Object] leadImpedance: protected float leadImpedanceThreshold: private float leadOneInPin: private enum leadTwoInPin: private enum leadOneOutPin: private enum leadTwoOutPin: private enum maxVOut: private float comPort: private uint8_t txRegister: private uint8_t rxRegister: private uint8_t
setLeadPins([enum]): protected void getLeadPins(): protected [enum] setMaxVOut(uint16_t): protected void getMaxVOut(): protected uint16_t setTxRxReg([uint8_t]): protected void getTxRxReg(): protected [uint8_t] voltageTest(float): protected float getCardiacEvents(): public Object clearCardiacEvents(): private void getLeadImpedance(): protected float getBatteryStatus(): public enum

Communications() extends Pacemaker
i_CommIn: [16bytes] vRaw: uint16_t f_Mmarker: uint16_t o_CommOut: uint8_t baudRate: uint32_t
connectDCM(): private bool sendEGM(): private bool initEGM(): public void recieveDeviceInfo(): protected [string] transmitDeviceInfo([string]): private void

Sense() extends Pacemaker
chambersSensed: private enum activityResponse: private enum magnetInPlace: private bool activityThreshold: private enum maxSensorRate: protected uint16_t
setChambersSensed(enum): protected void getChambersSensed(): public enum setActivityResponse(enum): protected void getActivityResponse(): public enum setMagnetInPlace(bool): protected void getMagnetInPlace(): public bool measureLeadImpedance(): protected void measureBatteryVoltage(): protected void setActivityThreshold(enum): protected void getActivityThreshold(): protected enum

Pace() extends Sense
pacingState: private enum pacingMode: private enum hysteresis: private Boolean hysteresisInterval: private uint16_t lowrateInterval: private uint16_t vPaceAmp: private uint16_t vPaceWidth: private uint16_t VRP: private uint16_t maxHeartRate: private uint8_t baseHeartRate: private uint8_t
setPaceMode(enum): protected void getPaceMode(): public enum setPaceState(enum): protected void getPaceState(): public enum setHysteresisInterval(uint16_t): protected void getHysteresisInterval(): public uint16_t setLowRateInterval(uint16_t): protected void getLowRateInterval(): public uint16_t setvPaceAmp(uint16_t): protected void getvPaceAmp(): public uint16_t setvPaceWidth(uint16_t): protected void getvPaceWidth(): public uint16_t setVRP(uint16_t): protected void getVRP(): public uint16_t setMaxHeartRate(uint8_t): protected void getMaxHeartRate(): protected uint8_t setBaseHeartRate(uint8_t): protected void getBaseHeartRate(): protected uint8_t

3.2 Utility Classes

The following tables outlines the public, private and protected methods making up each class defined above in section 3.1. Note that the *Sense* and *Communications* classes extend the *Pacemaker* class allowing them to inherit the properties defined in the Pacemaker class. The *Pace* class extends the *Sense* class in order to inherit properties of both Pacemaker and Sense. This allows information to be hidden in an appropriate class but made accessible without storing in multiple locations through getter and setter methods.

Class 1: Pacemaker()

This class stores information essential to the operation of a generic pacemaker. It includes variables describing the status of the battery, location of GPIO ports and memory addresses for TxRx I^2C operations. The methods and variables in this class are limited in scope and provide only a support framework on which other classes are able to operate within.

Method Name	Return Type	Description	Next Action (If action event triggered)
setLeadPins([enum])	void	Sets values for Lead(x)InPin,Lead(x)OutPin based on hardware GPIO requirements	None
getLeadPins()	[enum]	Accesses values of Lead(x)InPin,Lead(x)OutPin	None
setMaxVOut(float)	void	Sets maxVOut variable to maximum safe pace amplitude based on battery capacity	None
getMaxVOut()	float	Gets vale of maxVOut	None
setTxRxReg([uint8_t])	void	Sets hex memory locations of Tx and Rx registers storing serial buffer	None
getTxRxReg()	[int8_t]	Gets array of Tx / Rx register locations	None
voltageTest(float)	float	Takes arg min pace amplitude and increases voltage until ERM registers P-QRS-T sequence. Returns this voltage.	None
getCardiacEvents()	Object	Return object containing all stored cardiac events in EEPROM	None
clearCardiacEvents()	void	Erases EEPROM containing stored cardiac event data	None
getLeadImpedance()	float	Gets value of leadImpedance	None
getBatteryStatus()	enum	Uses values of batteryVoltage and replaceBatteryVoltage to	None

		determine battery status {BOL,ERN,ERT,ERP}	

Class 2: Sense()

This class contains variables and methods that are responsible for dealing with sensor input to the pacemaker device. The module hides information concerning sensor thresholds and configuration. Methods within this class interface with peripheral sensors through inherited GPIO port information and access / store information for use by other modules.

Method Name	Return Type	Description	Next Action (If action event triggered)
setChambersSensed(enum)	void	Takes chambers sensed as enum type {NONE, ATRIUM, VENTRICLE, DUAL} and sets value of private variable chambersSensed	None
getChambersSensed()	enum	Returns current value of chambersSensed	None
setActivityResponse(enum)	void	Takes activity response as enum type {NONE, TRIGGERED, INHIBITED, DUAL} and sets value of private variable activityResponse	None
getActivityResponse()	enum	Returns current value of activityResponse	None
setMagnetInPlace(bool)	void	Sets value of boolean var magnetInPlace.	None
getMagnetInPlace()	bool	Returns value of magnetInPlace that can be used to determine if diagnostic magnetism source in place	None
measureLeadImpedance()	void	Used internally to sense and set value of variable leadImpedance following measurement.	If impedance measured greater than leadImpedanceThreshold, set vPaceAmp in pace class to maxVOut. Log event.
measureBatteryVoltage()	void	Used internally to sense battery voltage and set value of batteryVoltage variable following measurement	If battery voltage below thresholdBatteryVoltage, enter power-saving state.
setActivityThreshold(enum)	void	Sets value of activityThreshold {V-Low, Low, Med-Low, Med, Med-High, High, V-High}	None

getActivityThreshold()	enum	Returns value of activityThreshold	None

Class 3: Communications()

This class is responsible for using serial communication protocols in order to send and receive data to and from the DCM application. It includes data structures to store and transmit EGM data as well as send and receive critical device information e.g. deviceID, implantDate, etc.

Method Name	Return Type	Description	Next Action (If action event triggered)
connectDCM()	bool	Contains required serial authentication procedures. Returns true on successful connection.	None
sendEGM()	bool	Method begins transmitting EGM phase and amplitude data over serial when called	None
initEGM()	void	Configures serial connection to send 16 byte EGM packets to DCM interface	None
transmitDeviceInfo()	[string]	Sends device info {deviceID, implant date, lead implant date, battery votage, cardiac events,…,etc} to DCM for interrogation	None
receiveDeviceInfo([string])	void	Configures serial connection to receive & store device data.	None

Class 4: Pace()

The pace class contains variables and methods that are used to produce the prescribed external pacemaker functionality required by the patient. It contains methods for setting the desired pacing mode, pacing parameter values and other variables enabling the desired therapeutic effect to be achieved within the patient. This class uses its inheritance from both the sense and pacemaker classes in order to interface with the attached leads and onboard sensors.

Method Name	Return Type	Description	Next Action (If action event triggered)
setPaceMode(enum)	void	Takes desired pace mode as enum per Generic NBG code {VVI, VOO, AOO, DDDR, etc}	Calls setChambersSensed(enum) and setActivityResponse(enum) from Sense() class.

getPaceMode()	enum	Returns current value of pacingMode	None
setPaceState(enum)	void	Takes pace state as enum type {PERMANENT, TEMPORARY, PACE_NOW, MAGNET, POWER_ON_RESET}, sets value of pacingState	Triggers appropriate methods in Pace() and Pacemaker() classes
getPaceState()	enum	Returns current value of pacingState	None
setHysteresisInterval(uint16_t)	void	Sets vale of hysteresisInterval which defines an additional delay interval used when value of hysteresis is True	None
getHysteresisInterval()	uint16_t	Returns current value of hysteresisInterval	None
setLowRateInterval(uint16_t)	void	Sets value of lowrateInterval that specifies maximum delay after a ventricle pace without a spontaneous sense or another pace	None
getLowRateInterval()	uint16_t	Returns current value of lowrateInterval	None
setvPaceAmp(uint16_t)	void	Sets value of vPaceAmp variable representing current amplitude of ventricle pacing output voltage	None
getvPaceAmp()	uint16_t	Returns current value of vPaceAmp variable	None
setvPaceWidth(uint16_t)	void	Sets value of vPaceWidth private variable representing current width of ventrical pace signal (ms)	None
getvPaceWidth()	uint16_t	Returns current value of vPaceWidth	None
setVRP(uint16_t)	void	Sets the value of variable VRP, duration of ventricular refractory period	None
getVRP()	uint16_t	Returns current value of VRP variable	None

setMaxHeartRate(uint8_t)	void	Sets the value of maxHeartRate later used to set upper frequency of pacing	None
getMaxHeartRate()	uint8_t	Returns current value of maxHeartRate	None
setBaseHeartRate(uint8_t)	void	Sets value of baseHeartRate later set to set minimum safe frequency of pacing for particular patient	None
getBaseHeartRate()	uint8_t	Returns current value of baseHeartRate	None

3.3 UI Class Methods

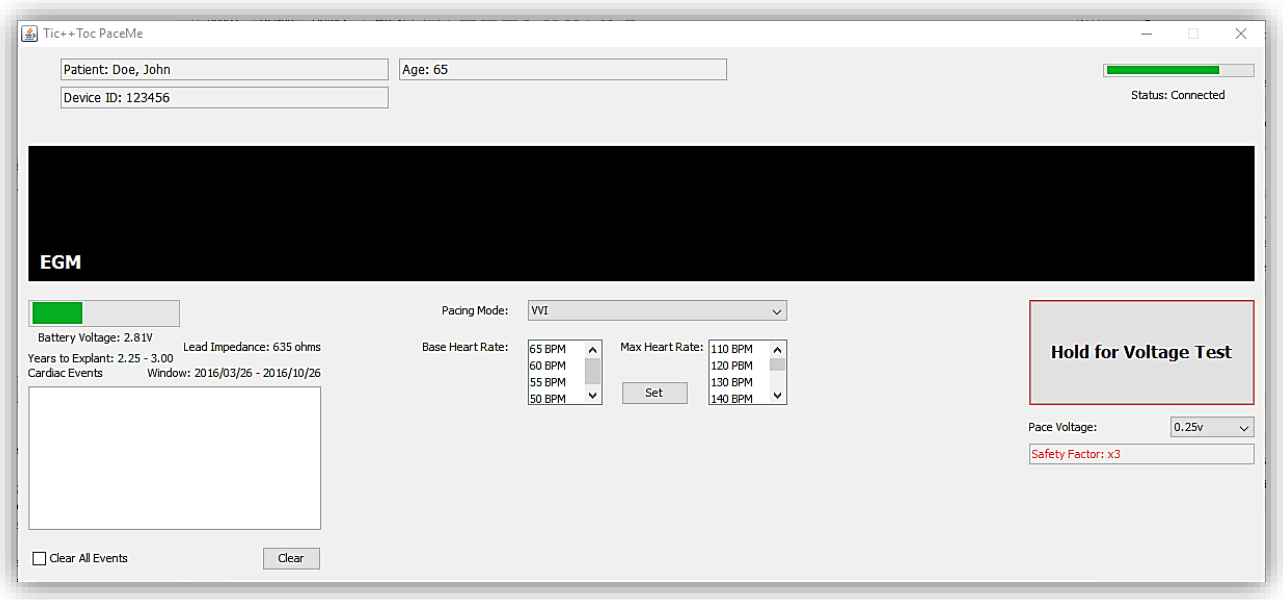

Figure 1 - Prototype DCM Interface

The user interface displayed above in Figure 1 - Prototype DCM InterfaceFigure 1 shows an approximate layout for the computer-driven DCM to be used by qualified doctors and nurses. The interface is designed to show important information such as patient info, device ID, communication status, and battery voltage in a clean, easy to read manner. The DCM is designed to take advantage of methods and parameters in the pacemaker code in order to customize functionality for individual patient needs while maintaining information hiding constructs. All information received and transmitted by the DCM is routed through the Communications() class in the pacemaker code effectively making this class and it's methods an intermediary between the user input and the safety-critical state variables controlling the pacemaker's overall behavior. Changes to the look and functionality can be expected as more pacemaker functionality is added, however this intermediary behavior is expected to remain unchanged.

3.4 Design Requirements Likely to Change

Requirement	Reason for Potential Change
Logged Detail of Cardiac Events Detected	Detailed logs of cardiac events may be kept for diagnostic purposes, however, given an abundance of such events, detail may need to be decreased in order to preserve storage space.
p_vPaceAmp & p_vPaceWidth	As scar-tissue generates over-top of pacemaker leads, resistance between leads subject to change. Applied voltage to induce ventricular contraction may need to be changed accordingly.
Base Heart Rate	Depending on patient age / level of physical activity, resting base heart rate should be customizable.

3.5 Design Decisions Likely to Change

Design Decision	Reason for Potential Change
Appearance and features offered on the User Interface	In the future, because of the relative ease with which software can be changed, features may need to be added or removed from the UI.
Checks on whether a value is in appropriate range are not implemented at this point.	In order to minimize risk to patients and maximize safety of the implanted device, safety checks will be written as software development progresses to ensure values changed in the device by doctors or other medical staff are within a safe operating range as outlined in the requirements.
Data structures responsible for holding communications data between pacemaker and DCM are pre-declared arrays of fixed-size	When EGM data is transmitted in software testing, size of data-structures may need to be amended as number of stored points increases in practise. Provisions for dynamic arrays & vectors may also be added.