Appendix A

Interaction Tile

The interaction tile consists of the following components:

- ATMEGA168 micro controller with Arduino[1] boot-loader ;
- Arduino mini-USB serial adapter[2]
- ACS 13.56MHz RFID reader[3] with RF SOLUTIONS external antenna[4];
- multi-colour LED's;
- accelerometer;
- vibration motor;
- piezoelectric speaker;
- magnetic switches.

Software running on the Arduino micro controller communicates to a PC using the serial-over-USB protocol. The wiring diagram of the Interaction Tile is available in Figure A.1. The RFID reader is connected to the PC through USB separately, and uses the PC/SC communication protocol for card readers.

Software written in Python runs on the PC and handles the events generated by the Interaction Tile. It inserts events and data into a triple store (SIB) and queries it when information is needed. A reasoner (Pellet[5]) is used to reason about the inserted low-level events in order to infer higher-level results. When a user establishes a connection, two `NFCEnterEvent` events (generated by the RFID reader) by two different devices not currently connected, will result in a new `connectedTo` relationship between the two devices. Because `connectedTo` is a symmetric relationship, the reasoner will automatically infer that a connection from device A to device B means that device B is also connected to device A. Since `connectedTo` is also an irreflexive property, it is not possible for a device to be connected to itself. A `generatedBy` relationship is also created between the event and the smart device that generated it, along with a timestamp and other event metadata.

[1] http://www.arduino.cc
[2] http://arduino.cc/en/Main/MiniUSB
[3] http://www.acs.com.hk/index.php?pid=product&id=ACR122U
[4] http://nl.farnell.com/jsp/search/productdetail.jsp?sku=1304031&CMP=i-bf9f-00001000
[5] http://clarkparsia.com/pellet/

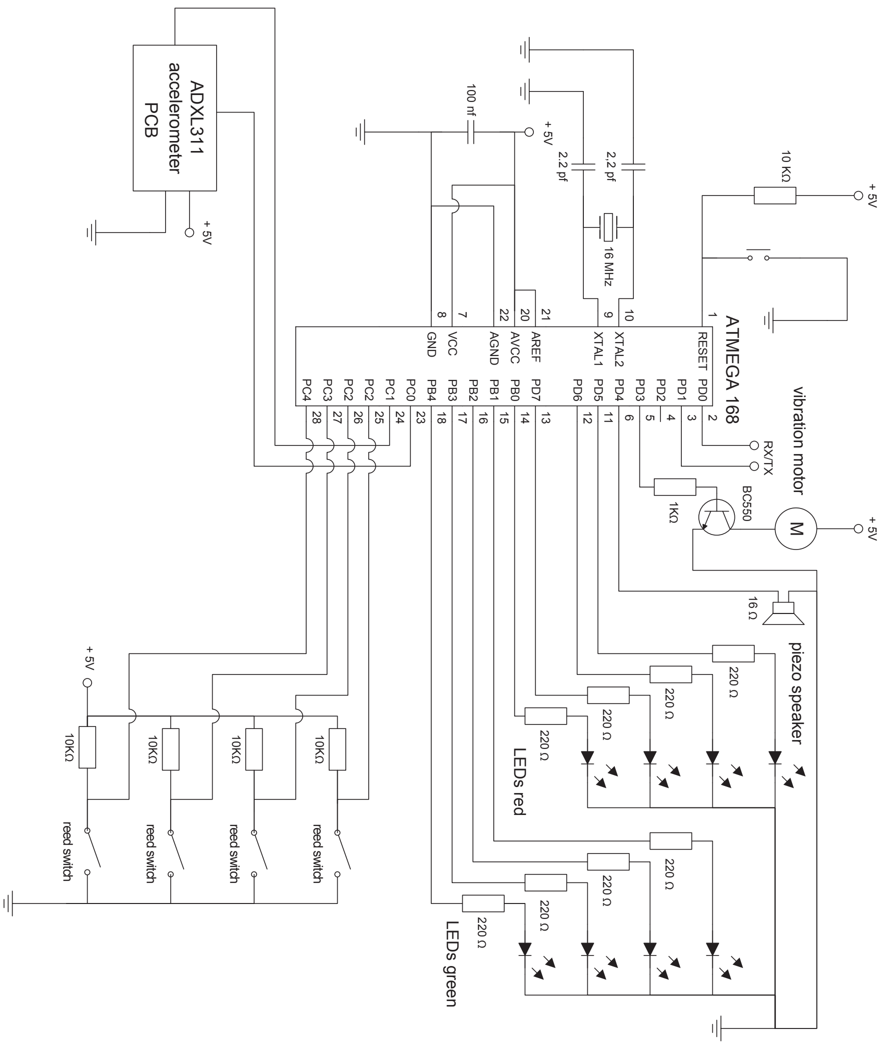

Figure A.1: Circuit drawing of the Interaction Tile hardware

Code examples of inserting triples into and querying triples from the SIB can be found in Listings A.1 and A.2 (code by Gerrit Niezen):

```
def addEvent(eventType, tagID, position):
    """Adds new event of *eventType* with metadata (**inXSDDateTime**,
        **hasRFIDTag** and **hasPosition**) to smart space"""
    t = RDFTransactionList()
    u1 = ns + "event" + str(uuid.uuid4())
    t.setType(u1, ns + eventType)
    #t.add_literal(u1, ns + "generatedBy", ns + self.deviceID)
    dt = datetime.now()
    xsddt = '"'+dt.strftime('\%Y-\%m-\%dT\%H:\%M:\%S\%z') + '"^^<http
        ://www.w3.org/2001/XMLSchema#dateTime>'
    t.add_literal(u1, ns + "inXSDDateTime", xsddt)
    t.add_literal(u1, ns + "hasRFIDTag", "\"" + tagID + "\"^^<http://
        www.w3.org/2001/XMLSchema#string>");
    t.add_literal(u1, ns + "hasPosition", "\"" + str(position) + "\"^^<
        http://www.w3.org/2001/XMLSchema#integer>")
    print eventType, ": Object ", tagID, " next to position ", position
    ts.insert(t.get())
    return u1
```

Listing A.1: Example Python code of inserting information

```
def showConnections():
    """
    Determines which devices are currently next to the interaction tile
         by querying the smart space,
    and an internal array storing the positions relative to the tile.

    Sends a response to the interaction tile (using ``sendToTile()``
    on whether a connection exists or whether a connection is possible
         .)
    """
    global rfid

    #Count number of devices
    devices = 4 - rfid.count("")
    print "Number of devices: ", devices

    for item in rfid:
        if item != "":

            if devices < 2:
                #If there is only one (or zero) devices, show that no
                     connections are possible
                sendToTile('n',rfid.index(item))

            #If the device isn't connected to anything, it won't be
                 found and set in the search below,
            #so at the end it should be set to "Possible"
            alreadySet = False

            device = getNameFromTag(item)

            qs = node.CreateQueryTransaction(smartSpace)
            result = qs.rdf_query([((device, ns + "connectedTo", None )
                 ,"uri")])
            for connDevice in result:
                print device.split("#")[1], " is connected to ",
                     connDevice[0][2].split("#")[1]

                #Is connDevice next to tile?
                for connItem in rfid:
                    connName = getNameFromTag(connItem)

                    if(connName == connDevice[0][2]):
                        #Yes, it's next to the tile, show as connected
                        sendToTile('c',rfid.index(item))
                        sendToTile('c',rfid.index(connItem))
                        alreadySet = True

            if (alreadySet == False) and (devices > 1):
                sendToTile('p',rfid.index(item))

            node.CloseQueryTransaction(qs)
```

Listing A.2: Example Python code of querying information

Code example of the Interaction Tile software is available in Listing A.3:

```
/** if more then one card exits at the same time, listen to the
  * accelerometer to detect an interaction event and send the
  * result of the interaction event over the serial line.
  * if only one card exits send the position of the exiting card.
  */
 void cardEvent()
 {
   boolean interactionEvent = false; //do we have an interaction event?

   int exitPos = -1;
   int prevExit;
   int reedEvent[4];

   for (int i = 0; i < 4; i++) {reedEvent[i] = digitalRead(reed[i]);}
       //store new reedvalues in an array
   for (int i = 0; i < 4; i++) {
     if (reedEvent[i] != reedState[i]) {  //compare arrays
       if (reedEvent[i] == LOW) {  //if card exits
         count ++;                              // add 1 to counter
         time = millis();                      // store current time
         if ((count > 1) && ((time - prevTime) < interval)) {
         // if more then one cards left and time difference
         // between two subsequent card exits is smaller
         // then the predefined interval
         // listen to interaction event
           for (int i = 0; i < 4; i++) {digitalWrite(ledR[i], HIGH);}
           while(!end) { // wait for an interaction to finish
             checkAccel();
             delay(10);
           }
           end = false;
           interactionEvent = true;
           count = 0;              // reset counter
           exitPos = -1;
         } else if ((time - prevTime) < interval) {
           time = prevTime; //equalize time and prevTime
         } else {
           exitPos = i;
         }
         prevTime = time; //prevTime should store the "current" time
       }

       if (reedEvent[i] == HIGH) { //if a card enters the field, write
           the position plus 10
         timeEnter = millis();
         count1 ++;
         if ((count1 > 1) && ((timeEnter - prevTimeEnter) < interval))
             {
           count1 = 0;
           //timeEnter = prevTimeEnter; //prevTime should store the "
               current" time
         } else {
           prevTimeEnter = timeEnter; //prevTime should store the "
               current" time
           Serial.write(i+10);
           Serial.write('E');  // followed by serial event marker
```

```
          }
        }
      }
    }
    if (exitPos >= 0) {
      Serial.write(exitPos);
      Serial.write('E'); // "E" is the serial event marker
    }
    // if we have an interaction event, send the result over the serial
        line
    if (interactionEvent) {
      // not sure about this
      //Serial.write(prevExit+20);
      //Serial.write('E');
      if (toggle == true) {
        Serial.write('C');  // connected!
        Serial.write('E');  // "E" is the serial event marker
      } else {
        Serial.write('D');  // disconnected!
        Serial.write('E');  // "E" is the serial event marker
      }
    }
  }
```

Listing A.3: Code example of the Interaction Tile software

Appendix B

Connector

The Connector prototype is made out of four separate 3D printed pieces (as is shown in Figure B.1). The lower part and the top part of the Connector can be moved inward and outward serving as a two-way spring-loaded switch. The prototype packages all the necessary components into one integrated device which is wirelessly connected to a computer using a Bluetooth connection.

Connector contains the following main components:

- Arduino Stamp 02
- Innovations ID-12 125kHz RFID reader
- SparkFun BluetoothMateGold
- 8 3mm bi-colour LEDs
- Two switches
- 3.7V LiPo battery (850 mAh)
- 5V DC to DC step-up VPack PCB

A wiring diagram of the hardware is available in Figure B.3.

The software running on the Arduino micro-controller communicates via serial over Bluetooth to a PC that runs the KP software (written in Python). It also uses the SoftwareSerial library[1] for serial communication with the ID-12 RFID reader, using normal digital I/O pins. The code for communicating with the ID-12 RFID reader was based on work by Martin Rädlinger[2] and[3].

The software on the Connector reads the RFID tags and sends the tags-IDs to the Connector KP. The KP then queries the SIB to lookup the tag-ID and find the matching smart object. When two smart objects are selected, the KP determines the connection possibilities and communicates the result back to the Connector, which handles the appropriate feedback. When a user connects or disconnects two devices, this event is communicated to the Connector KP, which in turn inserts or removes a `connectedTo` relationship between the two smart objects.

[1]http://arduino.cc/en/Reference/SoftwareSerial
[2]http://blog.formatlos.de/2008/12/08/arduino-id-12/
[3]http://www.arduino.cc/playground/Code/ID12

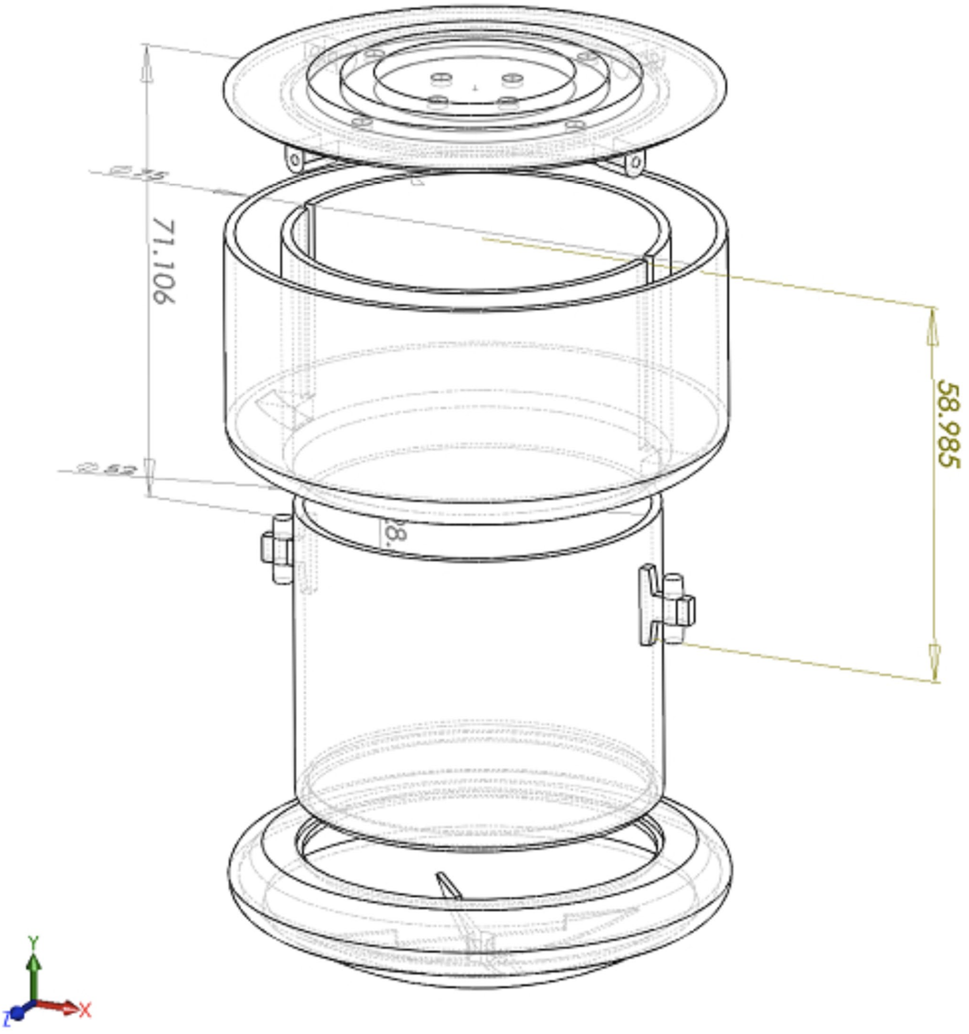

Figure B.1: Exploded view of the Connector

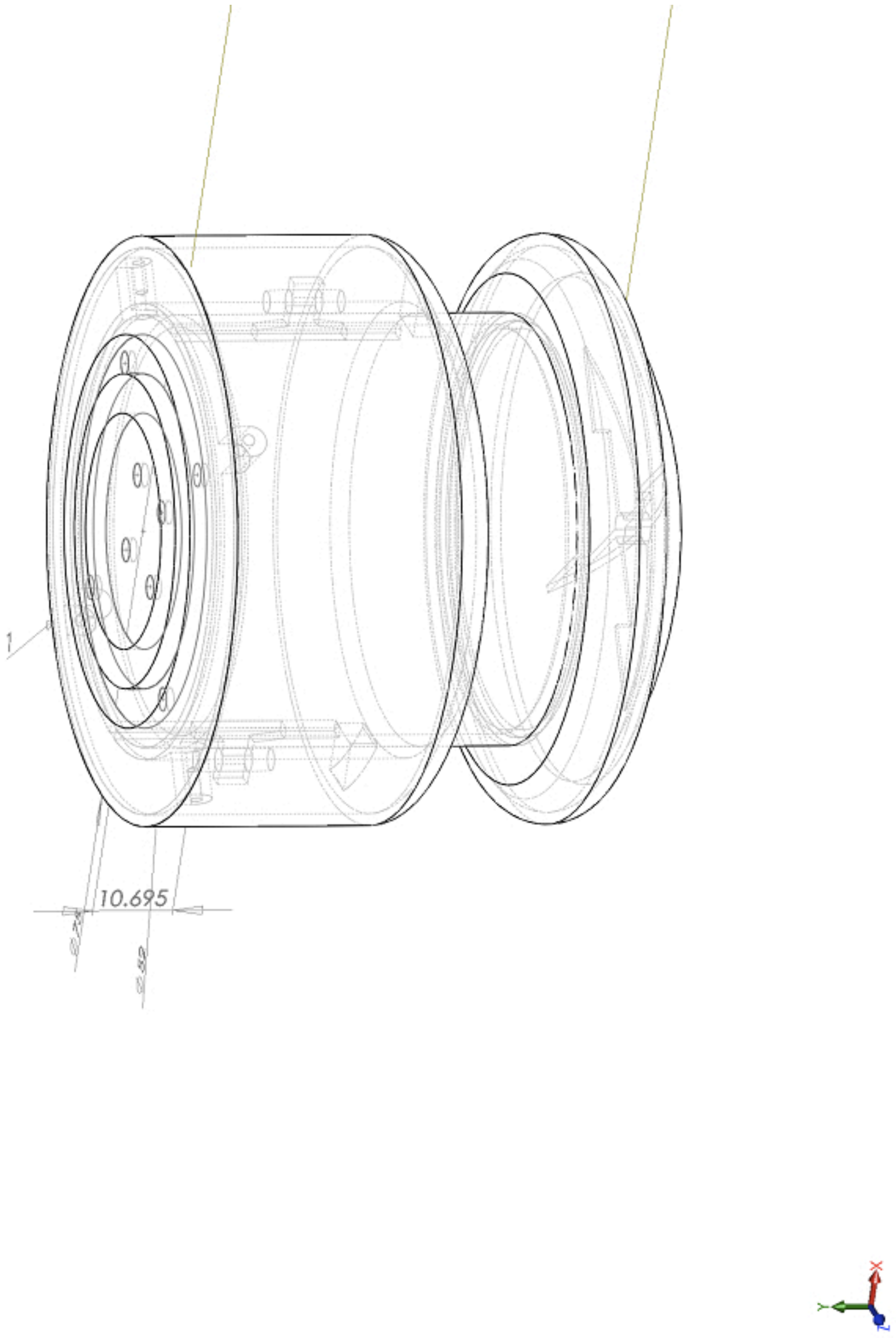

Figure B.2: A wire-rendering of the Connector's 3D CAD model

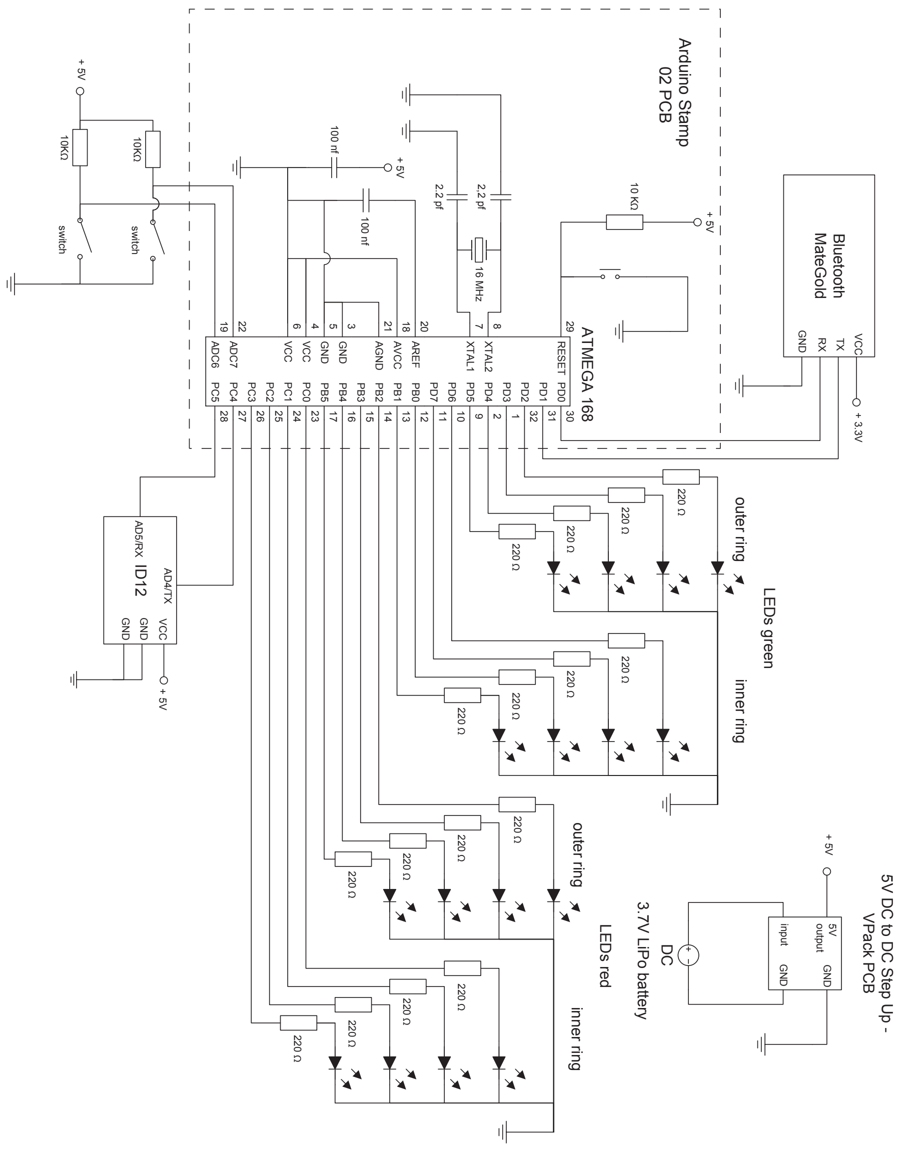

Figure B.3: Circuit drawing of the Connector hardware

Code example of the Connector software can be found in Listing B.1:

```
if (rfidPresent && firstDevice && !secondDeviceConfirmed)
  {
    deviceIDConfirmed = true;
    firstDevice = false;
    Serial.write("C");
    Serial.write("1");
  }
  else if (rfidPresent && secondDevice && !secondDeviceConfirmed)
  {
    deviceIDConfirmed = true;
    secondDeviceConfirmed = true;
    firstDevice = true;        // reset firstDevice
    secondDevice = false;      // reset secondDevice
    Serial.write("C");
    Serial.write("2");

    // wait while reasoning
    //we use analogue I/O ports since we're out of digital ones
    reasoning = true;
    SW_DOWN_MAP = analogRead(SW_DOWN);
    SW_DOWN_VAL = map(SW_DOWN_MAP, 0, 1023, 0, 1);
    SW_UP_MAP = analogRead(SW_UP);
    SW_UP_VAL = map(SW_UP_MAP, 0, 1023, 0, 1);

    while (reasoning)
    {
      serialComm();
      SW_DOWN_MAP = analogRead(SW_DOWN);
      SW_DOWN_VAL = map(SW_DOWN_MAP, 0, 1023, 0, 1);
      SW_UP_MAP = analogRead(SW_UP);
      SW_UP_VAL = map(SW_UP_MAP, 0, 1023, 0, 1);
      if ( SW_DOWN_VAL == 1 || SW_UP_VAL == 1)
      {
        reasoning = false;
        reset();
      }
      delay(100);
    }
  }
  else
  {
    //turn off LEDs
    if (ledRingState == 1) {} //except when LEDs are blinking
    else
    {
      LEDsInner(0);
      LEDsOuter(0);
    }
  }
```

Listing B.1: Code example form main loop of the Connector software

Code example of the ConnectorKP software can be found in Listing B.2:

```
def canConnect(device1, device2):
 global notFound

 if(notFound):
  return 'N'

 connectionPoss = 'N'

 qs = node.CreateQueryTransaction(smartSpace)
 result = qs.rdf_query([((device1, ns + "canConnectTo", None ), "uri")
     ])
 print result
 for canConnDevice in result:
  if(device2 == canConnDevice[0][2] and device1 != device2):
   #Devices can connect
   connectionPoss = 'P'
   break

 node.CloseQueryTransaction(qs)

 return connectionPoss
```

Listing B.2: Code example for checking whether two devices are connected