Redundant Archival Preservation System

Niko Hildenbrand, Zach Halvorson, Sara Fagin, Huda Irshad

Introduction

The Redundant Archival Preservation System was designed as a redundant environment monitoring system for use in museum settings, specifically for the preservation of paper and ink based materials such as books, paintings, or scrolls. The system also implements Marzullo's Algorithm for sensor fusion, triplicate redundancy to increase the accuracy of measurements, and short term failure sensors to maintain resiliency. As part of the modeling for this system, the physical continuous dynamics of the environment were simulated, including the air temperature, humidity, and light exposure as lux. The current system controls a set of alert LEDs to notify employees working to maintain proper conditions. Future improvements can be made to these LED control signals to convert the control signals into controls for AC units, heaters, humidifiers, dehumidifiers, and shades to control light exposure.

Design Functionality, Communication, and Implementation Details

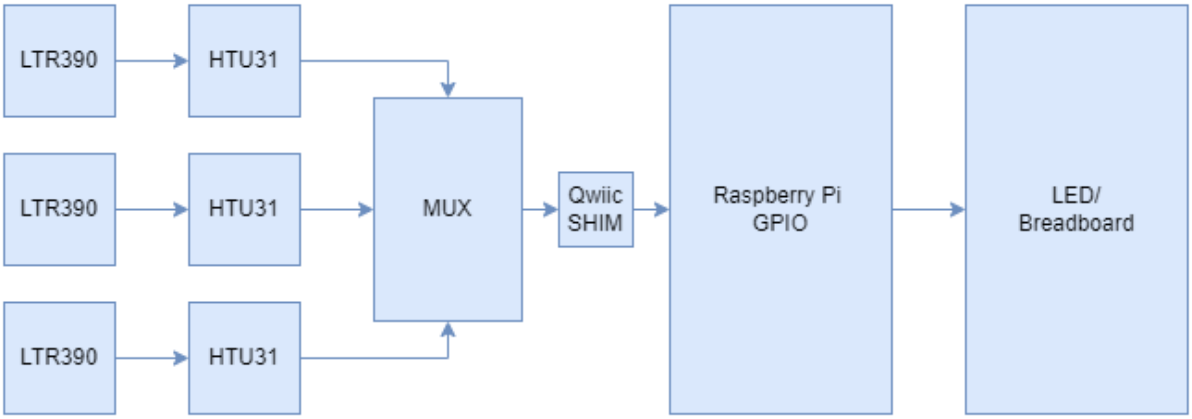

Figure 1: High Level Hardware Model

Three sets of LTR390[6] and HTU31[7] breakout boards are connected to the Raspberry Pi over i2c. Since each series of sensors use the same i2c addresses, only one series can be read

at a time. To do this, an 8 channel i2c mux was used. The mux allows us to choose which sensor series to connect with by enabling and disabling the mux channels. The Raspberry Pi acts as the central hub of this design. Taking in the readings from all three sensor series through the mux, running it through the sensor fusion python script, then outputting the results in the form of a CSV log file and the GPIO pins. In a real-world application, the GPIO pins would control actuators to respond to environmental triggers. For the purposes of demonstration, actuators have been replaced with a series of LEDs to indicate which alert has been triggered.

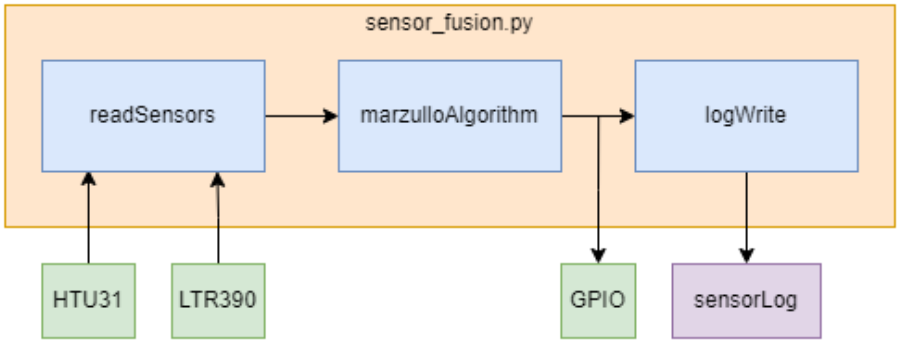

Figure 2: High Level Software Model

Sensor Readings from the HTU31 and LTR390 are read in by the readSensors function, 1 series at a time. The up/down status of the sensors is also checked at this time. The readings are then split into intervals based on the +/- precision error of the individual sensors. If at least 2 sensors are up, the intervals are then passed to the marzulloAlgorithm function, where the smallest common intersection of the intervals is determined and returned. Marzullo's Algorithm was introduced by Keith Marzullo in 1984, initially designed to estimate time accurately from a set of separate time sources with various levels of noise [1]. In our use case, the algorithm is used to improve the precision error of our sensor reading result. This is done by taking a set of intervals, formed by adjusting a sensor reading by the upper and lower bounds of the sensor's known precision error. These intervals are sorted in ascending order based on the lower value of the interval, then each interval is compared against the others to check if they overlap. If they

don't, they're skipped, if they do, a new interval is formed from the highest left value, and the lowest right value, to represent the intersection. The new interval is compared against the remaining intervals. After all of the comparisons are complete, the interval formed from the greatest quantity of overlaps is returned. In the event of a tie, the smallest interval is chosen. Figure 3 demonstrates an example of Marzullo's Algorithm as performed on a set of lux values:

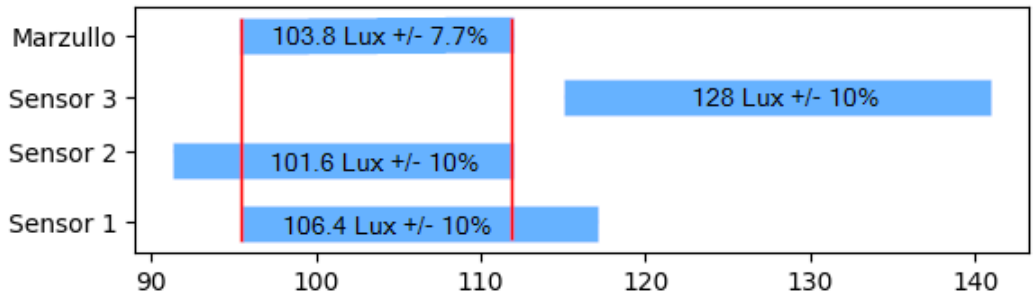

Figure 3. Demonstration of Marzullo's Algorithm

For less than 2 sensors, Marzullo's Algorithm is skipped and the initial intervals are used for any comparisons. Once the final intervals are determined, these values are then compared against the environmental thresholds stated in the specifications. This includes temperature, humidity, lux, and sensor status. For each threshold triggered, a specific LED is set to high, to indicate that an actuator should be turned on to react to the current environmental state, or that a sensor may be faulty. Additionally, a log file is written to every cycle, each file containing a day's worth of sensor readings and status alerts.

Specification and Modeling

The requirements for temperature, humidity, LUX, and alerts were specified for the museum setting. Humidity of artifacts in a museum were defined to be within the range of 35% to 55% with an ideal humidity level of 45%. Additionally, the humidity should not fluctuate +/- 10% within an hour [2]. With this research, the following specifications were made:

- Humidifier is on with full power when humidity level is less than or equal to 35%
- Humidifier is on with half power when humidity level is less than or equal to 40% and greater than 35%
- Dehumidifier is on with full power when humidity level greater than or equal to 55%
- Dehumidifier is on with half power when humidity level greater than or equal to 50% and less than 55%
- Neither, humidifier and dehumidifier, are on when the humidity level is greater than 40% and less than 50%
- When the fluctuation of the humidity level within an hour interval exceeds 10% then an alert must be sent

Temperature of artifacts in a museum were defined to be within the range of 20 degrees Celsius to 22 degrees Celsius [2]. With this research, the following specifications were made:

- Heating system is on with full power when temperature is less than or equal to 20.0℃
- Heating system is on with half power when temperature is less than or equal to 20.5℃ and greater than 20.0℃
- Cooling system is on with full power when temperature greater than or equal to 22.0℃
- Cooling system is on with half power when temperature greater than or equal to 21.5℃ and less than 22.0℃
- Neither, cooling and heating system, are on when the temperature is greater than 20.5℃ and less than 21.5℃

Lux of artifacts in a museum were limited to 200 lux in a single instance and 1000 lux hours in a day [3]. With this research, the following specifications were made:

- The shutters (covering of the artifact) will shift to a closed position when the lux sample in a single instance exceeds 200 lux
- The shutters (covering of the artifact) will shift to a closed position when the lux sample in a single instance exceeds 1000 lux hours within a given 24 hours period

With multiple sensors, the concern of accurate or outlier data at any moment arises since the system will react with undesired outputs. To avoid this, temperature range from the sensors utilizes Marzulo's algorithm and sensors' status are monitored with the following requirements:

- When only one sensor is alive, the temperature,humidity and lux values given to the controller are that of the remaining sensor
- When more than one sensor is alive, the temperature,humidity and lux values are calculated using Marzullo's algorithm
- Alert will be produced when a sensor, or multiple sensors fail
- Alert will be produced when a specific sensor remains down for 3 cycles in a row

The implemented UPPAAL model was designed to simulate a system/controller model that interacts with the environment model and vice versa, given the specific initial conditions of the environment model, as shown below in Figure 4. The High/Low is referring to the range endpoints of the data calculated by Marzullo's algorithm for the temperature, humidity, and lux. The soft/hard is referring to the power of the actuators for the controller of the system for temperature and humidity. For example, if the humidity controller system outputs soft dehumidifier as the value 1 then that is telling the environmental model that the dehumidifier is on with half power.

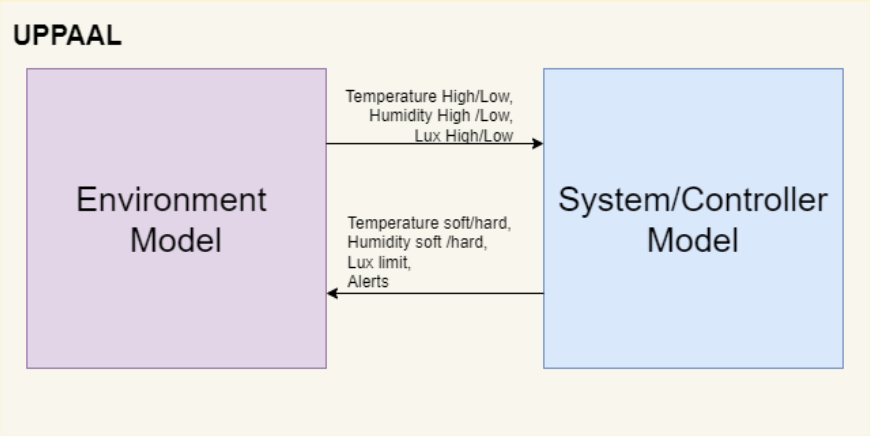

Figure 4: Environment and system/controller model as a feedback composition design.

The temporal model of the system described in Figure 5, located below, is sequential. The model is controlled with a series of broadcast signals to trigger the appropriate FSMs for the sequential sequence needed for the environmental model and the system/controller model. As shown in Figure 5, specific models will react synchronously with one another while others will react sequentially.

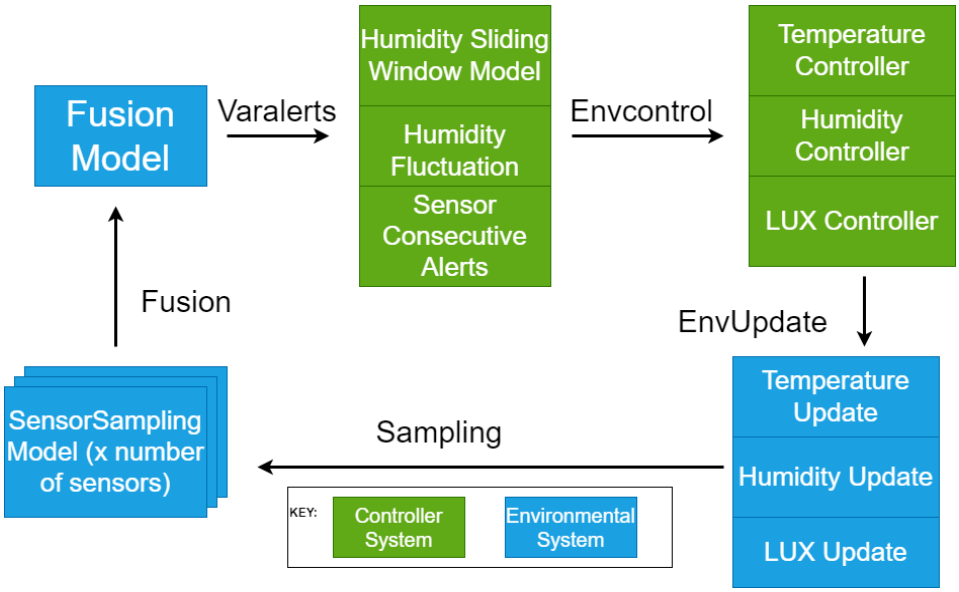

Figure 5: The sequential order of broadcast signals to generate the sequential sequence of model reactions needed to simulate the UPPAAL model

The purpose of the update models for temperature, humidity, and lux is to update the environmental variables according to the state of the controller models, which provides new values for the controller models to react to. This relationship can be described as a feedback composition.

The sensor status used to determine how Marzullo's will be executed is simulated by randomly selecting a 1 or 0 for the sensor status in the Sensor Sampling Model. The requirements for Marzullo's algorithm are implemented and occur each time a reaction of the Fusion model is triggered.

The alerts implemented in the simulation are variables that hold a 0 or a 1 value depending on if the requirements are met. The alerts for the sensor status are read to determine how Marzullo's algorithm will run because the requirements of Marzullo's algorithm execution is dependent on the sensor status. The alerts for the humidity alert is used as a notice to the intended user to be aware that the event of a significant humidity fluctuation within the 60 minutes has occurred.

Software and Hardware Implementation

When looking at the three Adafruit HTU31 temperature and humidity sensors used in our design, they have sustained performance even when exposed to extreme temperature [-40° to 125°C] and humidity [0%RH to 100%RH].In regards to temperature specifications, this sensor has a typical accuracy of ±0.2°C from -0~100°C. In regards to humidity specifications, the sensor has a typical accuracy of ±2% at 20% to 100% relative humidity in a temperature range from 0-70C. In addition, the humidity response time is five seconds and the supply voltage ranges from 3V to 5.5V. This sensor also uses I2C address 0x40 or 0x41, has a hardware reset pin, and an address-select pin to have two on one I2C bus.

Next, when looking at the three Adafruit LTR390 UV Light Sensors, the operating temperature ranges between -40 - 85 C. This part includes two photodiodes, one filtered for UV (325nm peak) exposure, and one filtered for visible light (550nm peak), where the visible light photodiode is used to produce lux values. These parts have I2C interface to run on Arduino or Python microcontrollers The CircuitPython library has routines to derive the UV Index value from the UV measurements along with calculating the ambient light Lux levels. When looking at the pinouts of these parts, it has the following: VIN, 3V0 and GND. It also includes I2C logic including SCL, SDA, and INT pins.

Analysis and Justification

The analysis and justification of the implementation meeting the correct specifications, was completed in 2 seperate ways. One with testing the UPPAAL built model, and the other through physical testing of our combined hardware. Both the UPPAL and physical model testing have the same specifications, along with the same set of tests to justify those specifications are met. All specifications have been tested and verified in UPPAAL and the physical environment.

Shown below are two separate tables, both of which help with the justification on the implementations meeting the above specifications. The first table shows the conversions of actual values, to UPPAAL modeled values for simulation purposes. As we mention in the technical challenges section of this report, the lack of support for floating point numbers in guard conditions in UPPAAL, forced the scaling of values, which has been outlined below.

	Actual Value	UPPAAL Modeled Value
Temp 1	20.4 ℃	20400
Temp 2	21.6 ℃	21600
Temp 3	19.9 ℃	19900
Temp 4	22.1 ℃	22100
Humidity 1	39%	3900
Humidity 2	51%	5100
Humidity 3	24%	2400
Humidity 4	56%	5600
LUX	200	20000

The second table, shown below, illustrates the UPPAAL queries of the specification mentioned, and the starting conditions that these queries were tested in. All queries tested below were verified with our model and confirmed that all specifications have been met.

English Language Query	UPPAAL Modeled Query	Starting Condition
If 1 set of sensors goes down for 1 cycle, can the system still function and provide readings for preservation.	E<> (sensor_fail_alert[0] == 1 && LUXout > 0 && TEMPout > 0 && HUMout > 0) || (sensor_fail_alert[1] == 1 && LUXout > 0 && TEMPout > 0 && HUMout > 0) || (sensor_fail_alert[2] == 1 && LUXout > 0 && TEMPout > 0 && HUMout > 0)	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If 1 set of sensors goes down for 3 cycles, can the system still function and provide readings for preservation.	E<> (sensor_offline[0] ==1 && LUXout > 0 && TEMPout > 0 && HUMout > 0)|| (sensor_offline[1] ==1 && LUXout > 0 && TEMPout > 0 && HUMout > 0) || (sensor_offline[2] ==1 && LUXout > 0 && TEMPout > 0 && HUMout > 0)	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If 2 sets of sensors go down for 1 cycle, can the system still function and provide readings for preservation	E<> (sensor_fail_alert[0] ==1 && sensor_fail_alert[1] == 1) || (sensor_fail_alert[0] ==1 && sensor_fail_alert[2] == 1)|| (sensor_fail_alert[1] ==1 && sensor_fail_alert[2] == 1) && LUXout > 0 && TEMPout > 0 && HUMout > 0	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If 2 sets of sensors go down for 3 cycles, can the system still function and provide readings for preservation	E<> (sensor_offline[0] ==1 && sensor_offline[1] == 1) || (sensor_offline[0] ==1 && sensor_offline[2] == 1)|| (sensor_offline[1] ==1 && sensor_offline[2] == 1) && LUXout > 0 && TEMPout > 0 && HUMout > 0	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If 1 sensor in a set goes down, can system deal with this imbalance	E<> TEMP[0] == -1 || TEMP[1] == -1 || TEMP[2] == -1 || HUM[0] == -1 || HUM[1] == -1 || HUM[2] == -1 || SLUX[0] == -1 || SLUX[1] == -1 || SLUX[2] == -1 && LUXout > 0 && TEMPout > 0 && HUMout > 0	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If temperature is less than 20.4 C, will heating system turn on with half power	E<> TEMPout <= 20400 && soft_heat == 1	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If temperature is greater than 21.6 C, will cooling system turn on with half power	E<> TEMPout >= 21600 && soft_cool == 1	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity

If temperature is less than 19.9 C, will heating system turn on with full power	E<> TEMPout <= 19900 && soft_heat == 1 && hard_heat== 1	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If temperature is greater than 22.1 C, will cooling system turn on with full power	E<> TEMPout >= 22100 && soft_cool == 1 && hard_cool == 1	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If humidity is less than 39%, will humidifier turn on with half power	E<> HUMout <= 3900 && soft_hum == 1	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If humidity is greater than 51%, will dehumidifier turn on with half power	E<> HUMout >= 5100 && soft_dehum == 1	int temperature=23000, humidity=6000, LUX=1000; //high temperature and high humidity
If humidity is less than 34%, will humidifier turn on with full power	E<> HUMout <= 3400 && soft_hum == 1 && hard_hum == 1	int temperature=21000, humidity=4500, LUX=2000; //perfect temperature and perfect humidity
If humidity is greater than 56%, will dehumidifier turn on with full power	E<> HUMout >= 5600 && soft_dehum == 1 && hard_dehum==1	int temperature=23000, humidity=6000, LUX=1000; //high temperature and high humidity
If humidity has an absolute change of greater than 10% per hour, does an alert get sent to indicate fluctuation	E<> (humFluxSum >= 10000 || humFluxSum <= -10000) && humAlert == 1	int humFluxSum = 15000 and int humFluxSum = -15000;
When Lux is over 200 Lux at one reading, do shutters shift to a closed position	E<> LUXout >= 2000 && LUX_limit == 1	int temperature=21000, humidity=4500, LUX=2000;
When Lux is over daily light exposure of 1000 lux hours, do shutters shift to closed position	E<> LuxSum >= 6000 && LUX_limit == 1	int temperature=23000, humidity=6000, LUX=1000, LuxSum = 7000; //high temperature and high humidity

Technical Challenges

One technical challenge when designing our model in UPPAAL was the lack of support for floating points in guard conditions and simulations, which after researching was discussed at the following sites [4,5].To solve this, we scaled each of the values by 10X or 100X or 1000X in order to work with integers as UPPAAL seems to currently require for FSM guards and simulations. This was a straightforward fix, but did introduce additional room for error when having to keep track of proper scaling across FSMs and other documents. UPPAAL, at least in 4.1.19 only implements an int16 type, where an int32 type would have allowed us to scale all of the values by a similar amount, instead of needing to do different scales for each value.

Additionally, when simulating random fluctuations between the sensors when sampling the environment, we quickly reached a state space explosion issue. For example, we first attempted to simulate a sensor sampling a hypothetical current lux of 1000, and designed it so that each sensor could produce a value between 900 and 1100 (+/- 10% for lux), independent of each other. Simulating the entire range of 200 lux values (900 to 1100), for each of the three sensors, already is a combined set of 8 million possible transitions, not even counting combinations with other FSMs that occur at the same time.

Division of Work

Zach built many of the UPPAAL FSM's (Temperature, Humidity, SequentialGen, Fusion, SensorSampling, SensorConsecAlerts), assisted in verifying the entire UPPAAL model, checked for errors in our overall simulation, implemented Marzullo's Algorithm in UPPAAL based off of Sara's work in Python, formulated the differential equations for setting the environmental dynamics in UPPAAL, managed ordering the parts, contributed to the presentations and final paper.

Sara coded the Python script based off of the agreed specifications, taking in the readings from the sensors over i2c, running them through Marzullo's Algorithm, and directing

the output to a series of LEDs. Also implemented sensor status checks to determine if sensors were down, and using this to determine how the remaining sensor results should be handled in relation to Marzullo's Algorithm. Additionally, output results to a log per the specifications. Assisted with the implementation of Marzullo's Algorithm in UPPAAL. Sara also designed the hardware layout and assembled the series of LEDs, and sensors with the Raspberry Pi. Wrote the README for the GitHub repository, providing instructions for the configuration and use of the Raspberry Pi Sensor hub.

Niko helped build out numerous FSMs in UPPAAL while also helping with verification of the entire UPPAAL model. He also helped edit and test the implementation of Marzullo's algorithm in UPPAAL . Additionally he formulated all system specifications and test cases in english and converted them to UPPAAL to ensure the correct design of the system. He also outlined all relevant info from the data sheets for hardware and software components to help structure the design of the physical system. Lastly, he helped contribute to all presentations, final demo and final paper.

Huda collaborated with Zach for initial design system controller FSMs and their dynamics, built several UPPAAL Models(Humidity Action, Temperature Action, LUX Action, Humidity Sliding Window, Humidity Fluctuation), contributed to the presentations and final paper. She helped edit and discuss rescaling of values to meet the range limits in UPPAAL.

Conclusion

The redundant controller system designed to monitor and react behaved as expected for the thermal dynamics implemented in simulation and when testing the physical system. The simulation was designed as a feedback composition of the environmental model and redundant controller model to observe the redundant controller system with realistic inputs as 'time' continued. The physical model was tested using external elements (humidifier, fan, flashlight, and hair dryer) to change the current environment faster to observe the outputs of the physical

system rather than depending on uncontrollable factors to test the system. Both the simulation and physical system behaved similarly for equivalent environments.

Please find our software files and more information at our GitHub repo:

https://github.com/sfagin89/RedundantSensorProject

Please find a recording of our presentation here:

https://youtu.be/ZMbAT5cT7FY

References

[1] Keith Marzullo Maintaining the Time in a Distributed System

[2] Appraisal and Preservation Resources for Books | The Art Institute of Chicago

[3] Light Exposure for Artifacts on Exhibition

[4] "Server connection lost" when using fint() · Issue #23 ·UPPAALModelChecker/UPPAAL-Meta

[5] Add Uppaal SMC metamodel. by ennoruijters · Pull Request #1 · uppaal-emf/uppaal

[6] LTR390 UV Light Sensor Data Sheet

[7] HTU31 Temperature & Humidity Sensor Data Sheet