Universidad de los Andes Colombia

Universidad de los Andes

Implementation and Validation of the XWare Smart Retrofitting middleware

David Sanchez-Londono
Giacomo Barbieri
Kelly Garces

2022

1 Background

The following is a supplementary document to the research paper "XWare: a Middleware for Smart Retrofitting in Maintenance" [1] (unpublished at the time of writing). Said work proposes *XWare* (eXperimental middleWare) as a low-cost middleware for smart maintenance applications that eases the implementation of the communication capability into legacy devices. The work overviews the reasoning behind the creation of XWare and the design process that was used. It also discusses the potential impact of XWare and potential future works. This supplementary document then goes into detail on the implementation and validation of XWare from a computer science perspective.

2 XWare Implementation

This section illustrates the implementation of XWare. As a hardware solution, Raspberry Pi[1] devices are used for both the gateways and the server. Raspberry Pi devices were selected for their low cost and high software flexibility. All code was programmed in Python 3.

2.1 XWare components

XWare is currently implemented in an environment that only uses legacy devices, and not M2M devices. This means that sensors are connected to gateways, which are then connected to a local server. Next, the components involved in the XWare operation are illustrated:

- *Operator*: user in charge of defining the data acquisition parameters for each sensor.
- *Sensor*: one of multiple sensors, from which samples are collected at a given frequency.
- *Gateway*: one of multiple Raspberry Pi devices. Their main function is to transmit samples collected from the sensor to the server through the M2M protocol.
- *Server*: main Raspberry Pi or Windows device. Its main functions are to integrate samples from multiple gateways and to build the final files that follow the XWare metamodel.
- *Application*: final destination for the acquired data.

Regarding the gateway, note that it must support different M2M protocols and fulfill the specifications in [1]. Therefore, the gateway is also in charge of connecting the broker of the selected M2M protocol (in this case MQTT) to the OM2M server.

[1] www.raspberrypi.org/

Figure 1 shows a diagram of the relationships between the different components, including the MQTT broker and the OM2M server inside the server component.

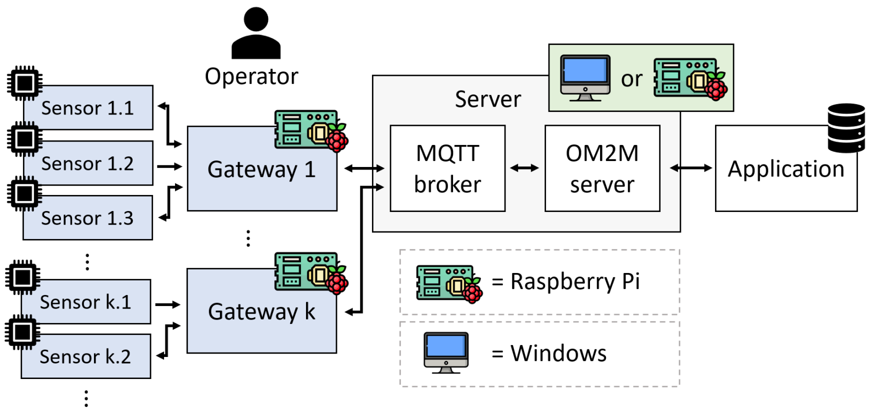

Figure 1: Components involved in the XWare environment.

2.2 Function instantiation

Having established the components and their fundamental relationships, the following functions must be instantiated.

- *Collect Samples*: samples from devices must be stored in local gateways.
- *Transmit Samples*: the collected samples must be sent to a local server, along with basic context information like the beginning time of the data collection and the sampling frequency.
- *Receive Samples*: the local server must receive samples from multiple devices.
- *Add Context*: samples must be enriched with the necessary context by translating them to the XWare metamodel. The metamodel is specified in [1]. Samples are also assigned a Sampling ID. This ID references a specific sampling that is defined in the application layer beforehand.
- *Transmit Samples* (*App.*): samples enriched with context must be sent from the local server to the application.

Figure 2 shows how MQTT, OM2M, and custom-made XWare components (both the metamodel and a Python library) are used to implement the different functions. Next, their implementation is illustrated.

First, sensor data are recorded in each gateway as part of the *Collect Samples* function. Then, Mosquitto MQTT (along with required Python libraries) is used to *Transmit Samples* from each gateway to the server. OM2M and its MQTT plugin then work in tandem to *Receive Samples* into the server. The custom-made XWare Python Library is then used to *Add Context* to the incoming samples. Finally, the

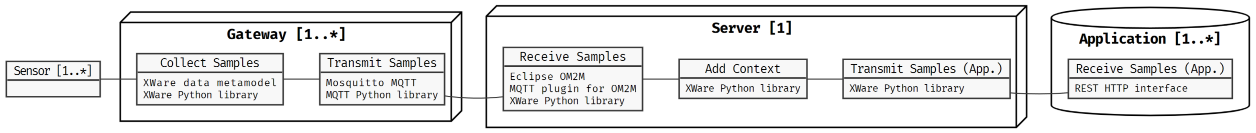

Figure 2: Functions that the XWare components perform, along with the technological solutions that each function uses. [1..*] indicates that there may be any number sensors, gateways and applications, while the server is a unique component.

same XWare library is used to *Transmit Samples* from the server to the application.

The version of XWare that is provided in the GitHub repository does not include this final *Transmit Samples* function; instead, samples with added context are stored locally for the user to utilize them. For testing and validation purposes, the XRepo 2.0 PHM educational information system was used as the application layer [2]. XRepo 2.0 is a big data information system that allows professors to share PHM sensor data with students and their peers. XRepo 2.0 was developed alongside the XWare metamodel, and thus the two are compatible without the need of a translator. The communication between the server and XRepo 2.0 occurs through a RESTful interface.

2.3 Functional sequence

The interactions between the components for the implementation of the functions are summarized in Figure 3 through a sequence diagram. The diagram shows the messages and data that are exchanged between the components illustrated in Section 2.1.

Initialization The sequence starts with an operator initiating the process by defining certain key parameters for each gateway and executing the Gateway Python code in each gateway. Figure 4 illustrates a few of these key parameters. Whereas, the complete list includes:

- F: The sensor sampling frequency.
- t: How many seconds should be sampled on each cycle.
- T: The time that should exist between one sampling cycle and the next - known as sampling period.
- The *conversion factor* between sensor values and real physical units; e.g. a conversion between mV and m/s^2 in an accelerometer.
- Device tag: indicates the gateway from which samples come from.
- Sensor tag: identifies the sensor to which a sample belongs

Identification Once the operator executes the Gateway Python code, the gateway identifies itself with the server. This involves creating *OM2M applications and containers* where the gateway will transmit data.

Clock synchronization The initialization is followed by the gateway starting the data collection process. This involves sending an initial START message to the server, who in turns stores the starting timestamp for this data sampling cycle. Knowing a starting time and a sampling frequency allows the server to assign timestamps to each individual sample. This process removes discrepancies that may arise from having non-synchronized clocks between the gateways.

Data collection After the starting timestamp has been stored (shown as 'Start new clock' in Figure 3), the gateway begins collecting data from the sensor with a given sampling frequency (F) for a given sampling time (t). This data collecting process occurs cyclically for a given time period (T) (see Figure 4). Note that only the sample values are collected, not the sample times.

Once the samples of a given measurement have been collected in the gateway, they are compiled in a CSV file; this file only contains sensor values, without times or context data. The CSV file is then sent along with an associated JSON file which does include context data (namely, the previously mentioned key parameters) to the Local Server through MQTT and OM2M. Finally, the data from the CSV and the JSON are joined into a single XWare metamodel file. This requires adding timestamps to each sample value, converting each value using the proper conversion factor, identifying the tags (Device, Sensor) for each value, and appending the appropriate Sampling ID.

Transmission This step is not included in XWare, but was instead used for the validation of the middleware. Once a set of samples is enriched with the necessary context, it is sent to the XRepo 2.0 information system. This enrichment involves converting data to the XWare metamodel.

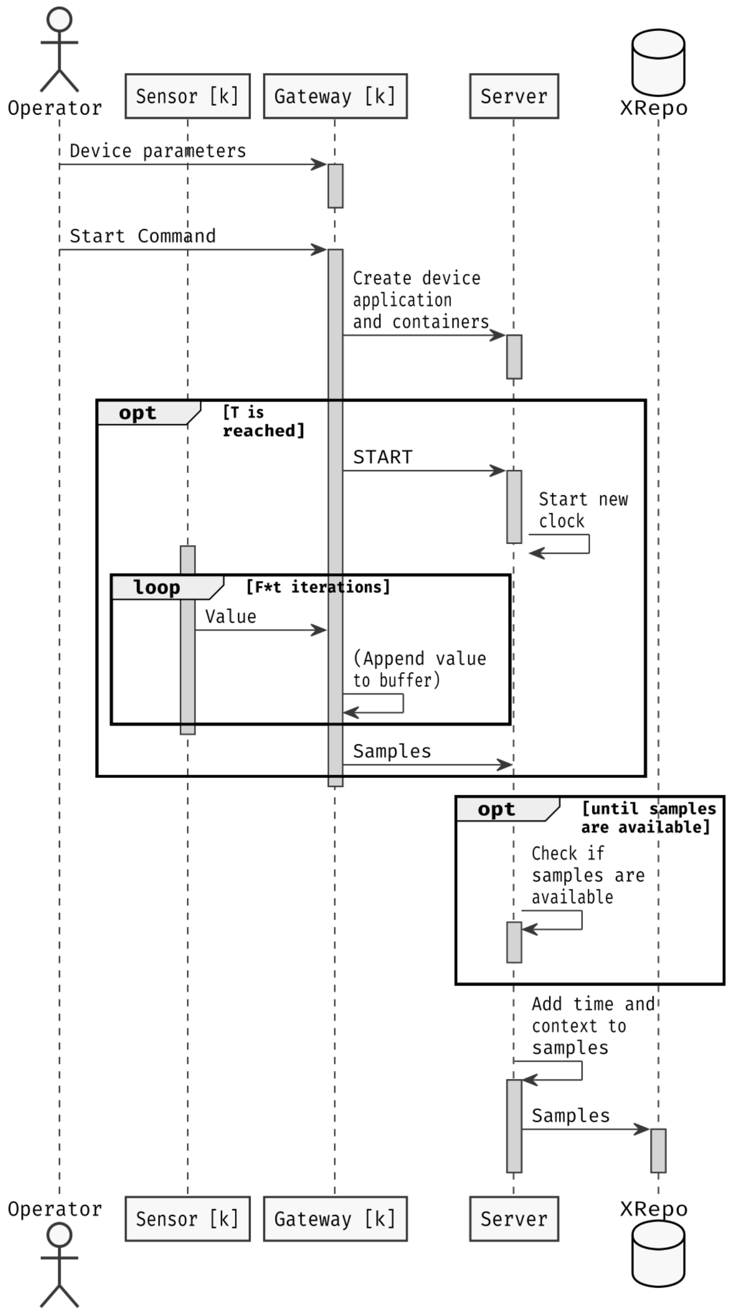

Figure 3: Sequence diagram of how the XWare components interact.

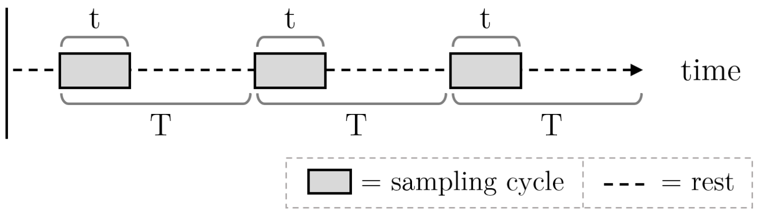

Figure 4: Multiple sampling cycles show the meaning of the t and T parameters in the sample collection process.

3 Verification

This section illustrates how the operation of XWare was verified. This validation is illustrated in Section 3.1, and includes a) ensuring sample data integrity, b) measuring the performance of the implemented architecture, and c) verifying the multiple gateway functionality. Whereas, the fulfillment of the requirements identified in [1] is shown in Section 3.2.

3.1 XWare verification

To verify the correct operation of the proposed XWare architecture, all the necessary software components were installed and tested on two separate Raspberry Pi devices. All tests were performed with a *simulated data acquisition*, with data taken from the educational shaft unbalance dataset found in [3]. To simulate the acquisition, the gateway read a value from this dataset every $1/F$ seconds.

3.1.1 Data integrity verification

Objective: The goal of this verification is to ensure that the integrity of sensor data is not lost when they are sent to the application layer. Retaining data integrity would mean that any analysis performed would give the same results whether data were acquired directly from the sensor or from the application layer.

Methodology: The following steps are required for this verification:

1. Simulate a data acquisition process on the XWare gateway, while making sure that these acquired data are **not** deleted from the gateway.
2. Make the acquired data follow the entire XWare process, moving to the server and then to XRepo.
3. Download the acquired data from the XRepo platform.
4. Manually recover the acquired data from the gateway, adding timestamps to each value and correcting the sensor values.
5. Extract relevant PHM features from both the original gateway data and the data obtained from the XWare-XRepo workflow.
6. Compare the original data and the XWare data by visually contrasting them in the frequency domain and by comparing extracted relevant PHM features.

Features belonging both to the time and frequency domains were selected for the comparison:

- *RMS*: Root mean square of the acceleration signal in the time domain.

- *Peak*: Peak amplitude near the motor frequency (about 30 Hz for this dataset) in the frequency domain.

Obtaining the same values for RMS and Peak in the original data and the XWare data would indicate that data integrity was maintained. A visual verification of the frequency domain would also strengthen this assumption.

A similar verification has already been done on XRepo [2, 4]. These previous tests involved manually uploading sensor data to XRepo and then downloading them. In this work, data are uploaded using the implemented XWare architecture. This means that any data discrepancies before and after the XWare-XRepo workflow would be due to XWare only, making this a valid test for verifying data integrity in XWare.

Results: This verification process was performed for two different samplings: one with no shaft unbalance, and the other with high shaft unbalance. Data corresponding to a single measurement file were analyzed from each dataset, taken both directly from the sensor and from XRepo. All data had a sampling frequency of $20kHz$ and a sampling time of $30s$. Figure 5 shows the visual comparison of both signals in the frequency domain. Table 1 shows the Peak and RMS features obtained from the original data and the XWare-processed data. The equivalence of signals indicates that data integrity is maintained.

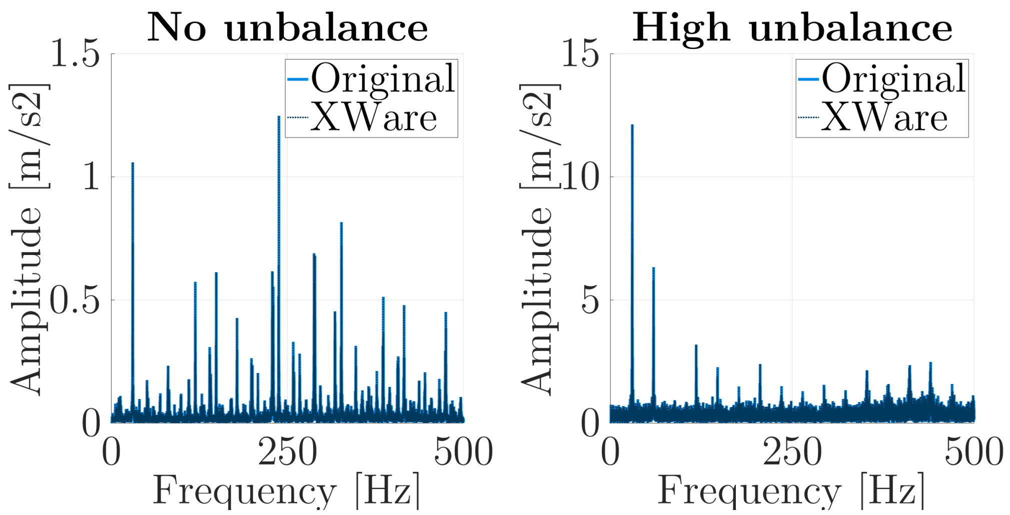

Figure 5: Comparison of two distinct signals (left and right) in the frequency domain, before and after passing through XWare and XRepo. Note how the signal is unchanged.

3.1.2 Performance test

In this section, the performance of the XWare architecture is quantified. All performance testing was run using the following hardware:

Table 1: Time and frequency domain features calculated before and after passing through XWare and XRepo. Note how the features are unchanged.

Dataset	No unbalance		High unbalance	
	Peak	RMS	Peak	RMS
Original	1.059	5.147	12.128	41.817
XWare	1.059	5.147	12.128	41.817

- *Gateway*: Raspberry Pi Model 3B. Quad core ARM CPU @1.2GHz, 1GB RAM, 8GB of flash storage, Debian-based Linux operative system[2].
- *Server*: Raspberry Pi Model 4. Quad core ARM CPU @1.5GHz, 2GB RAM, 16GB of flash storage, Debian-based Linux operative system.

Objectives: The objective of the test was to find whether a phase of the XWare architecture required more processing time than the others, suggesting optimizations to reduce *bottlenecking*. To do this, 110 measurement cycles occurred and the time that each specific phase required was recorded.

Methodology: The steps required for this verification are the following:

1. Define a desired combination of sampling parameters (F, t) for the test.
2. Perform a cycle of data acquisition via the XWare architecture, using these parameters.
3. Measure the time elapsed from the start of data acquisition to the successful upload to XRepo.
4. Begin a continuous acquisition process with a time period T that will allow for each cycle to occur without overlapping. In this continuous acquisition, record the time taken by each step of the XWare-XRepo data flow.
5. After a sufficiently large number of acquisition cycles, calculate the mean time that each step takes.

By finding how long a given XWare step takes in average, the processing time of each step can be found and compared to other steps. The steps that were analyzed for this work are:

- *Sampling*: Time in which sensor data are being collected. It is considered to be exactly equal to t.
- *Flight*: The time between the end of data acquisition in the gateway and the moment said data arrive at the server. A bottleneck here would indicate that either MQTT or OM2M were not being used properly.

[2]www.raspberrypi.org/software

- *Enrichment*: The time it takes for the raw sample values to be enriched with timestamps and context. A bottleneck here would show that the CSV-creating process is not optimized.
- *Upload*: The time it takes for the enriched samples file to be uploaded to XRepo. This time only includes the upload of a file to the XRepo online server; any further processing done by XRepo is not included here. A bottleneck here would be mostly indicative of a slow internet connection

The mapping of the measured phases and the functions of XWare is shown in Figure 6.

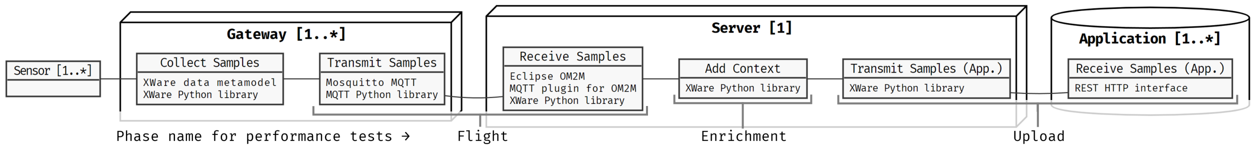

Figure 6: Functions that the XWare components perform, with the addition of the different phases that were considered for the performance tests.

Results: Following the given bottleneck test methodology, tests were performed with the parameters: $F = 20kHz, t = 1s, T = 30s$. 110 sampling cycles took place, resulting in the times found in Table 2. The uncertainty of each time was calculated using Equation 1. A visualization of the same times is provided in Figure 7.

XWare and XRepo were able to process each sample cycle in an average of 6.29 seconds. As seen in Table 2, the phase that took the longest time was the "upload", which involved the process of transmitting data to XRepo. However, reducing this time resides within the scope of XRepo and not XWare. Even with that bottleneck, 1 second of 20kHz data acquisition could be collected (for example) every 10 seconds without straining the architecture. Bottleneck testing then allowed the authors to verify that the developed XWare code (along with the use of MQTT and OM2M) was sufficiently efficient without any glaring time sinks in the process.

$$error = \frac{3\sigma}{\sqrt{N}} \tag{1}$$

Table 2: Time taken for 20 kHz, 1s sampling cycles to finish each stage of the XWare-XRepo data flow. The uncertainty was calculated considering a population of N=110 and the error on Equation 1. All values presented in seconds.

Sampling	Flight	Enrichment	Upload	**Total**
1±0	0.654 ±0.014	0.810 ±0.019	3.826 ±0.280	**6.290 ±0.280**

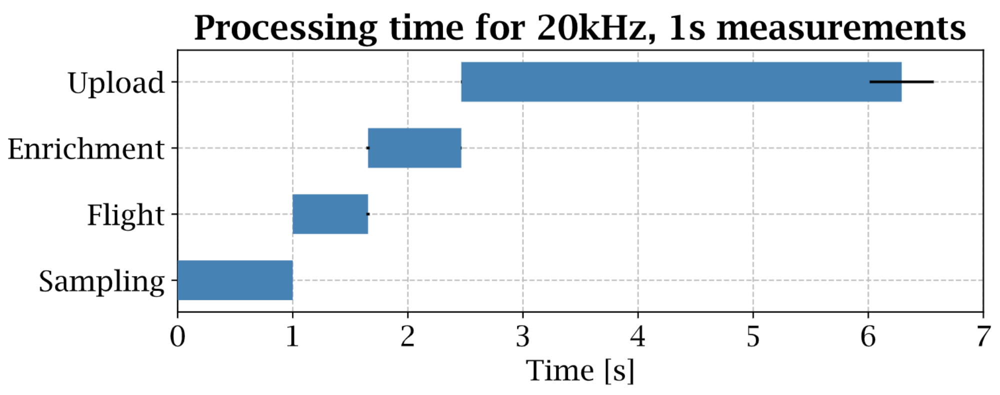

Figure 7: Time taken for 20 kHz, 1s sampling cycles in each stage of the XWare-XRepo data flow. Uncertainty values are taken from Table 2.

3.1.3 Multiple gateway validation

Next, the viability of using multiple gateways is verified. The tests were performed by running multiple instances of the XWare Gateway code in a single Raspberry.

Objectives: The main objective of the *multi-gateway* test is to verify that the implemented XWare code can manage multiple streams of incoming data.

Methodology: The test to verify that multiple gateways are supported is as follows:

1. Either initialize multiple Raspberry gateways with 1 instance of the XWare Gateway code on each, or execute multiple instances of the XWare Gateway code in a single gateway Raspberry; the latter route is followed for this work.
2. Assign different values of F, t, and T to each gateway.
3. Begin a continuous acquisition process on all the gateways, and record the time from the start of each data acquisition to the successful upload to XRepo.

Regarding the multiple-gateway acquisition, three gateways were executed in one Raspberry Pi for a duration of 50 minutes with the time parameters and results shown in Table 3. If the total time for a given measurement had been higher than its respective period T, the processing of measurements would overlap and the test would fail. The total time was always lower than the period T, indicating that the architecture is able to receive measurements from multiple gateways and enrich them with context simultaneously, uploading said samples to the application layer. It should be noted that the current XWare Server implementation can only process one data file at a time. This means that the multiple-gateway capabilities are made possible thanks to task queueing, rather than actual parallel processing of data streams.

Table 3: Parameters and total processing time for multiple gateway test.

	gateway0	**gateway1**	**gateway2**
Freq. F	1000 Hz	10 kHz	20 kHz
Time t	2 s	1 s	0.5 s
Period T	20 s	30 s	50 s
Population N	151	101	61
Total time	4.209 $\pm$0.038 s	6.278 $\pm$0.267 s	5.149 $\pm$0.317 s

3.2 Fulfillment of requirements

Next, the fulfillment of each of the input requirements defined in [1] is evaluated:

- *Metamodel*: Having surveyed various standardized PHM metamodels, a custom XWare metamodel was developed. This metamodel can store commonly-used PHM data which have been translated from multiple, potentially heterogeneous devices. This metamodel can also store with experimental and educational context, in the form of a JSON object that includes the sampling frequency F, time t, period T, the signal conversion factor, and the XRepo tags that reference all the context defined on that platform (such as the units of the different sensors, or the visibility of any given data to students).
- *Two-way communication*: The XWare architecture leverages the capabilities of MQTT and OM2M, which are M2M technologies that can both transmit messages *to* and *from* any connected device. This will allow any future developer that might want to implement Digital Twins in an educational context to base their work on XWare.
- *Cost*: All the technologies utilized in XWare are either free to use or have a low cost. Software wise, all pre-existing solutions used (MQTT, OM2M, Python, Debian operative system) can be obtained and utilized without licensing fees. Hardware wise, Raspberry Pi devices are known to be a low-cost computing solution; even if Raspberry Pi's are not available to a user, any device that runs a form of Linux should be able to act as a gateway or server. Finally, all custom-made XWare code will be shortly released in GitHub under an Open Source license alongside a usage manual.

References

[1] David Sanchez-Londono, Giacomo Barbieri, and Kelly Garces. "XWare: a Middleware for Smart Retrofitting in Maintenance". 2022 (Unpublished) (pages 1, 2, 6, 11).

[2] Nestor Romero, Rafael Medrano, Kelly Garces, Giacomo Barbieri, and David Sanchez-Londono. “XRepo 2.0: a Big Data Information System for Education in Prognostics and Health Management”. In: *International Journal of Prognostics and Health Management* –.Issue Pending (2020), pp. – (pages 3, 7).

[3] Giacomo Barbieri, David Sanchez-Londono, Laura Cattaneo, Luca Fumagalli, and David Romero. “A Case Study for Problem-based Learning Education in Fault Diagnosis Assessment”. In: *IFAC Proceedings Volumes* (*IFAC Papers-OnLine*) (2020) (page 6).

[4] Alfonso Ardila, Felipe Martinez, Kelly Garces, Giacomo Barbieri, David Sanchez-Londono, Andrea Caielli, Laura Cattaneo, and Luca Fumagalli. “XRepo - Towards an information system for prognostics and health management analysis”. In: *Procedia Manufacturing* 42 (2020), pp. 146–153 (page 7).