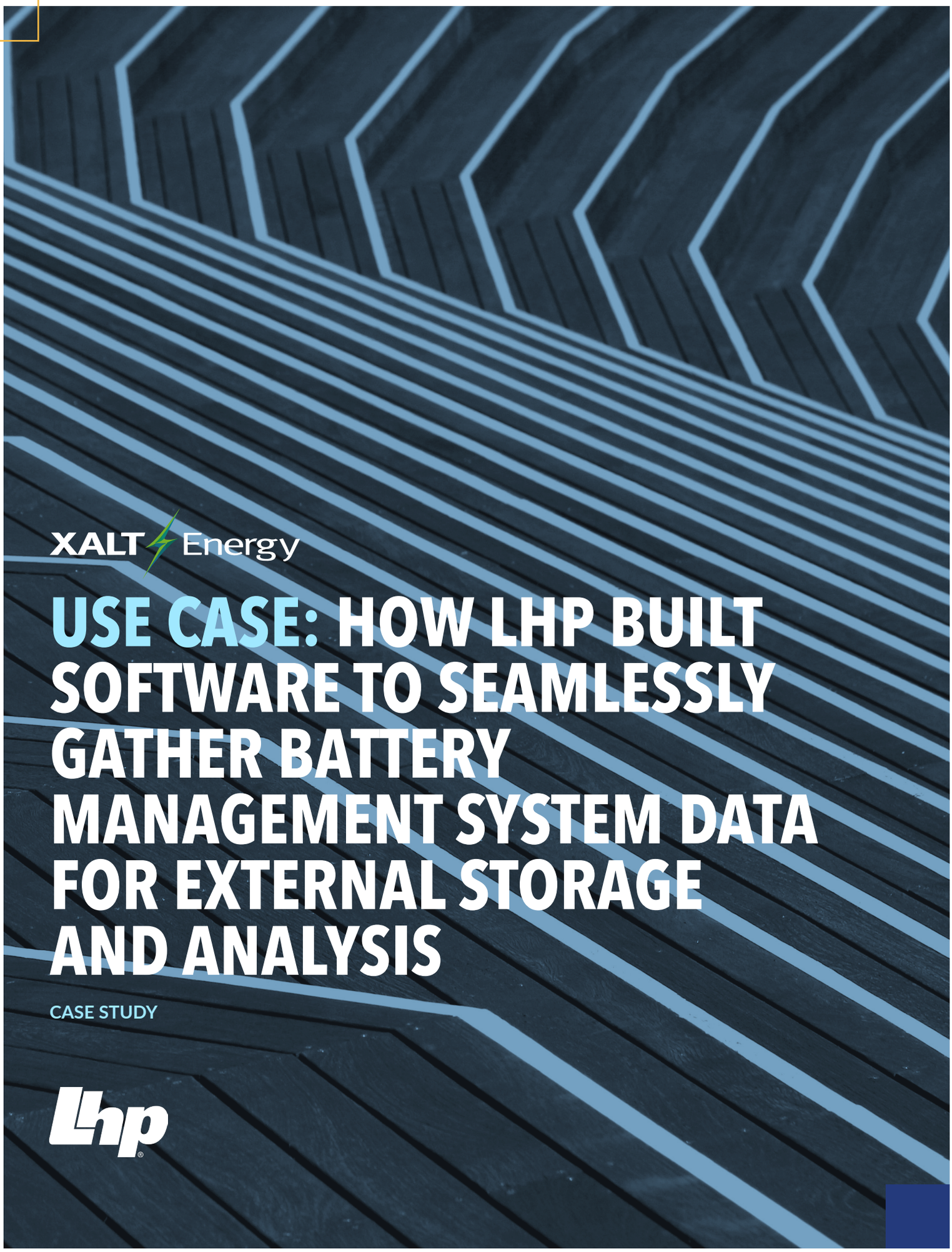

USE CASE: HOW LHP BUILT SOFTWARE TO SEAMLESSLY GATHER BATTERY MANAGEMENT SYSTEM DATA FOR EXTERNAL STORAGE AND ANALYSIS

CASE STUDY

CHALLENGES

XALT Energy is a manufacturer of energy storage solutions. They design and build lithium-ion cells, modular packs, and high-voltage, high-capacity multi-module arrays at their world-class manufacturing facility in Midland, Michigan. Their focus is on the growing demand for high-tech storage solutions in marine, commercial transportation, and specialty applications.

Modern energy storage solutions require precise, highly technical real-time battery management, in the form of systems that can capture, process, and utilize data about the battery system itself, as well as the components to which the battery is connected. This is accomplished with a series of interconnected devices working together.

The Battery Management System (BMS) provides the monitoring, balancing, and safety functionality for the system. It monitors battery parameters to control each cell in the battery pack. The main job of the BMS is to maintain the lithium-ion cells within their safe operating range regardless of the demands that are being placed on them. This means that the BMS must possess the ability to accurately predict operating situations that have the potential for creating over-voltage or under-voltage conditions, as well as high- or low-temperature conditions. The BMS must react quickly and safely to prevent those adverse conditions from occurring.

The capacity of each battery system differs from one cell to another. This difference increases with the number of charge/discharge cycles. So, the parameters being monitored by the BMS are constantly changing.

ABOUT THE PROJECT

Industy

- Energy Storage and Management

Company Name

- XALT Energy

Tools/ Technologies/ Skills

- Python, CAN, C#, Azure, SQL, Reverse Engineering from an existing code base, transfer/transform/ format data, vector CANalyzer, vector CAPL scripting

Goals of the Project

- Be able to collect and store data from the BMS to be used for analysis

Application Area

- The telematics device is in the field, and it transmits captured data to the data hub

The BMS consists of three parts:

- Master Control Unit (MCU): The MCU manages multiple parallel strings while providing a single and simple interface to the application controller. The MCU communicates with up to 24 SCUs (see below) to get data one section at a time. It then transfers this data back to the TCU (see below).
- String Control Unit (SCU): Each string communicates with the MCU via the battery system CAN (Controller Area Network) bus to transfer its data.
- Telematics Control Unit (TCU): The TCU communicates with the MCU to receive data by sending requests and listening for responses.

Before partnering with LHP, XALT had only built software to gather raw BMS data on a test machine. They wanted to gather data autonomously onboard the TCU itself, and then transform it into a usable format. LHP was engaged to create software that would gather the BMS data and then transmit it to an external server where it would be processed and stored. Additionally, they wanted all their data to be optimized for querying and analysis.

Customer Requirements

- Create a program to retrieve data from the BMS and compile this into an JSON-formatted logfile.
- Use the CAN bus and J1939 protocol to communicate with the MCU to get the required data.
- Parse the generated logfiles into SQL tables.
- Make the program customizable for different TCUs, and be able to customize how often the script should run or what addresses to send the requests to.

THE SERVICES LHP DELIVERED

LHP programmed a solution that runs on XALT's TCU. Using Python, the LHP team created a script that runs directly on the TCU to retrieve data from the MCU and SCUs. The data is collected by sending a request to the MCU. The MCU responds with the data for a specific section within an SCU. This requesting process is repeated to obtain data from all the sections and all the SCUs.

LHP employed a CAN bus and the J1939 protocol to communicate with the MCU. The LHP team created a console application in C# that is stored on an SFTP virtual machine. It parses the incoming logfiles into the SQL tables. Configuration variables are read from a table to allow the user to customize the options to suit their needs.

ROADBLOCKS AND SOLUTIONS

- What if the quantity of data in a section is greater or less than it is supposed to be? LHP added a checker to the log parser to confirm that the section contains the correct amount of data. If it does not, then the section is discarded. Since the data collected consists of rolling counts, the customer agreed that discarding the section data for one logfile was acceptable.
- Learning how to work on a remote setup: LHP utilized a remote connection to log into a test bench

setup. We had to test the program this way instead of working locally. LHP had to communicate with the customer to get their assistance if something needed to be done to the physical setup.

- How could we get the code to work with the correct libraries? If we tried to include external libraries, they would need to be installed on all of the TCUs in the field. Due to the limited capabilities of the TCU, we avoided using external libraries altogether. In one instance, LHP had to rewrite an entire function to get it working with the already included library.
- Compressing and decompressing the logfiles: The retrieval script is written in Python, but the log parsing script is written in C#. The retrieval script compresses the file before transferring it, and the log parser must decompress that file when it arrives. LHP wrote a BZip2 compression and decompression method in both languages, and it worked properly. But when LHP tried to combine the Python compressing and the C# decompressing, the program generated errors. To remedy the issue, LHP had to use a different library which they installed on the C# side since they have more control over that environment.

HOW LHP'S SERVICES BROUGHT VALUE TO THE CUSTOMER

- The solution LHP developed is easy to deploy to additional TCUs and can run in the background without any user intervention. Our scripts launch on startup; so even if power is lost, they will continue to run whenever power is restored.
- The log parser is easy to start and runs continuously in the background processing the files as they arrive. The data are parsed into separate tables for each section of the logfile. This allows for easy querying of the data when making an analysis.
- The configuration tables allow the user to edit variables without having to edit the code directly and then redeploy.
- LHP organized meetings with different teams, for our customer to discuss the system and share how things worked. This collaboration helped provide valuable information to both LHP and the customer. The customer was able to partner together with LHP to better understand their own system.

FEATURES

- The LHP solution is easily deployable via over-the-air updates to TCUs in the field, or through local updates using the USB drive.
- Our program runs locally and still functions even if the outside connection is lost. Logfiles will still be collected, but they won't be uploaded until a connection can be made.

RESULTS, ROI, AND FUTURE PLANS

There was no previous baseline for warranty fulfillment at XALT. The software LHP developed fulfilled a need that they previously did not have the capability to address. This resulted in a net-new return on opportunity for warranty recording.