SSC17-P2-19

DebriSat, Big Data, and How It Relates to Small Satellites

Joe Kleespies
University of Florida
Gainesville, FL USA; +1 813 361 1236
joeykleespies@ufl.edu

Norman Fitz-Coy
University of Florida
Gainesville, FL USA; +1 352 392 1029
nfc@ufl.edu

ABSTRACT

DebriSat is a hypervelocity impact (HVI) experiment that aims to generate new debris characterization data to update existing orbital debris breakup models. The DebriSat project is currently in the post-impact phase where debris fragments are collected, characterized, and recorded. The DebriSat debris categorization system (DCS) is a relational database-based big data management solution designed to handle the hundreds of thousands of unique data points produced by the DebriSat project. This paper discusses the development process of the DCS and describes how some techniques used in the system can be applied to similar small satellite missions to assist in the management of their big data challenges. Furthermore, this paper provides some performance analyses of various database engines and services that highlight potential pitfalls when dealing with such a large amount of data and metadata. Finally, this paper presents some of the lessons learned during the planning, development, and implementation stages of both the DebriSat project and the DCS and how these lessons can be integrated into future small satellite missions.

INTRODUCTION

Recent advancements in data storage technologies and the emerging prevalence of S-, X-, and now even Ka-band frequencies with data rates larger than 1 Mbps in the small satellite industry have led to an exponential growth in the amount of data small satellite missions are generating. Nowadays, small satellite missions aren't just collecting raw scientific data, these missions are also collecting a massive number of additional metadata such as precise timing information, location information, environmental measurements, and other data about the conditions in which scientific measurements are taken. This surge in the quantity of data and metadata coupled with the complexity of managing the relational links between data points poses a classic big data challenge that future small satellite missions will need to address.

DebriSat was not a small satellite. In fact, it was designed to be a representative model of modern LEO satellites and was the subject of a hypervelocity impact (HVI) test conducted in April 2014 to investigate/characterize debris fragments generated during on-orbit collisions.[1] Thus, DebriSat was never launched, never collected any in-situ measurements, and never sent a single line of telemetry. However, the DebriSat project is truly representative of the challenges encountered in a big data project. Currently, the project has recorded over 120,000 debris fragments and it is projected that, in total, over 250,000 debris fragments will be collected and characterized.[2] Each fragment is tagged, characterized, and recorded. Each record contains data fields such as mass, linear dimensions, average cross-sectional area, volume, shape, color and additional metadata fields such as timestamps, revision number, and up to 126 fields for high-resolution images. Altogether, the amount of data currently being generated by the project is massive; the expected size of the final data set is on the order of 14 TB. In this sense, the project parallels some of the recent and upcoming small satellite missions that aim to generate equally significant sets of data. DebriSat faces the big data challenge of managing the hundreds of thousands of data points being produced and the complex relationships between data points and metadata. In response, the DebriSat Debris Categorization System (DCS), a relational-database-based big data management solution, was developed to address the big data challenges of the project.[3]

This paper discusses the development process of the DCS and describes how some techniques used in the system can be applied to some small satellite missions to assist in the management of their big data challenges. This paper also provides some performance analyses of various database engines and services that highlight potential pitfalls when dealing with such a large amount of data and metadata. Finally, this paper presents some

of the lessons learned during the planning, development, and implementation stages of both the DebriSat project and the DCS and how these lessons can be integrated into future small satellite missions.

DESIGN OF THE DCS

After the DebriSat HVI test in April 2014, the project entered the post-impact phase. During this phase, debris fragments produced from the HVI test are to be collected, characterized, and recorded. It was clear from the beginning of the post-impact phase that some sort of database or data management system would be needed to facilitate this collection, characterization, and recording of debris fragments.

Requirements

Like any small satellite mission or systems engineering process, the design and development of the DCS began by establishing a set of preliminary requirements. These requirements, described in Table 1, applied not only to the DCS, but also to the post-impact characterization effort in general.

Table 1: Preliminary DCS Requirements

ID	Requirement Description
1	Facilitate entry and recording of identification, assessment, and characterization data for soft-catch foam panels used in the DebriSat HVI test.
2	Facilitate entry and recording of identification, assessment, and characterization data for debris fragments produced by the DebriSat HVI test.
3	Facilitate verification and validation of all recorded data.
4	Facilitate regular backups of all recorded data.
5	Secure access to all recorded data and allow only authorized users to add, modify, or verify recorded data.
6	Record all actions executed on recorded data, when these actions were executed, and who executed them.
7	Facilitate the centralized storage of all recorded data and the transfer of all recorded data between organizations.

Architecture Overview

To achieve the requirements outlined in Table 1, the general architecture shown in Figure 1 was devised. The DCS consists of a front-end user interface for user entry of identification, assessment, and characterization data for soft-catch foam panels and debris fragments and a set of back-end services to facilitate the recording, querying, and backup of entered data. The front-end user interface and back-end services are hosted on a private server located on the University of Florida campus. Recorded data is backed up to a secondary location on the University of Florida campus.

The front-end user interface is designed using PHP, HTML, and JavaScript to present easy-to-use forms and fields to facilitate the entry and validation of data to the end user. The end users, in the DCS's case, are the students collecting, characterizing, and recording soft-catch foam panels and debris fragments.

The back-end services consist of a web hosting service, database engine, and backup service. Microsoft's Internet Information Services (IIS) is used for the web hosting service to present the PHP, HTML, and JavaScript pages to the user on the front end.[4] Oracle MySQL and Microsoft SQL Server are used as database engines for the recording and querying of data.[5,6] Finally, Microsoft Windows Task Scheduler and VSS Backup are used for the backup service to periodically copy and backup all recorded data to a secondary location.

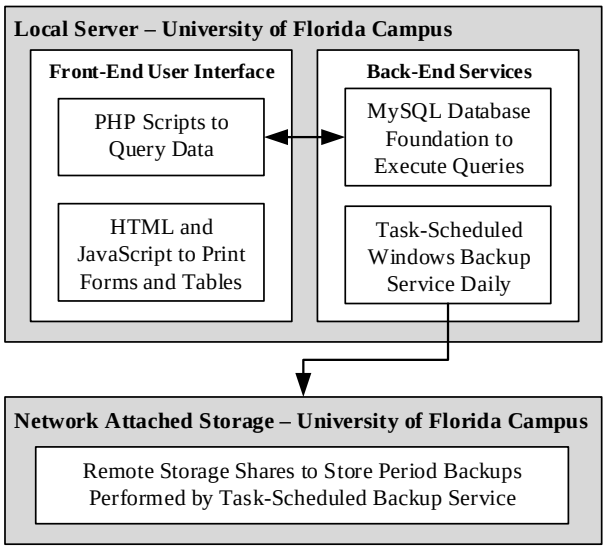

Figure 1: Overview of the DCS Architecture

Back-End Services

The selections for the back-end services were made before the development of the DCS front end. IIS, Windows Task Scheduler, and VSS Backup were all chosen for the web hosting and backup services because they are all self-contained and are tightly integrated within the Microsoft Windows Server environment. The Windows Server environment is supported by all the member organizations of the project and there is extensive documentation, support, and validation of these services. Thus, they were selected for the web hosting and backup back-end services.

The selection of a database engine was a bit more extensive. There are a multitude of different commercial database engines available. Oracle MySQL and Microsoft SQL Server (MSSQL) are two of the most popular engines. The MySQL database engine excels for development environments and smaller production

environments while MSSQL is designed for large-scale and enterprise production environments. Both database engines have a large set of extensions and add-ons that can be used to enhance their implementations. For example, the MySQL database engine is compatible with the PHPMyAdmin web-based management interface, which makes database management much more streamlined.[7] Similarly, the MSSQL database engine includes FILESTREAM functionality for advanced file system integration.[8] Both database engines are customizable to the intended application yet excel in different environments.

During the design and development of the DCS, the MySQL database engine was used. However, as DebriSat's data set has expanded in size and scope, the MSSQL database engine has been phased in to enable an alternative back-end database engine to address future scalability challenges. Currently, the DCS supports both the MySQL and MSSQL database engines.

Front-End Interface

The DCS front-end interface is responsible for facilitating the entry, modification, validation, and verification of all recorded data. The goal when designing the DCS front-end interface was to streamline this process of entry, modification, validation, and verification. The front-end interface is designed to include identification, assessment, and characterization data for each debris fragment and soft-catch foam panel and was designed in conjunction with the overall post-impact phase workflow, divided into detection, extraction, and characterization. This workflow, shown in Figure 2, produces the various identification, assessment, and characterization data for each debris fragment and soft-catch foam panel.

One of the guiding design principles for the DCS and the DebriSat post-impact effort is to minimize human error and maximize efficiency. Designing the DCS front-end interface in conjunction with the post-impact phase workflow enables very high user efficiency. To minimize human error, the DCS front-end interface is designed to restrict and validate user inputs during the initial entry and subsequent modification processes.

Furthermore, the front-end interface is designed to include external verification functionality as a last step in the post-impact phase workflow. Verification is performed by users who have not edited the debris fragment or soft-catch foam panel in question to provide a third-party unbiased verification of the recorded data.

Finally, the DCS front-end interface is designed to include inherent security mechanisms to allow only authorized users to view, add, edit, or verify recorded data. The front-end interface is also designed to include logging and auditing functions to ensure complete revision histories for each debris fragment and soft-catch foam panel.

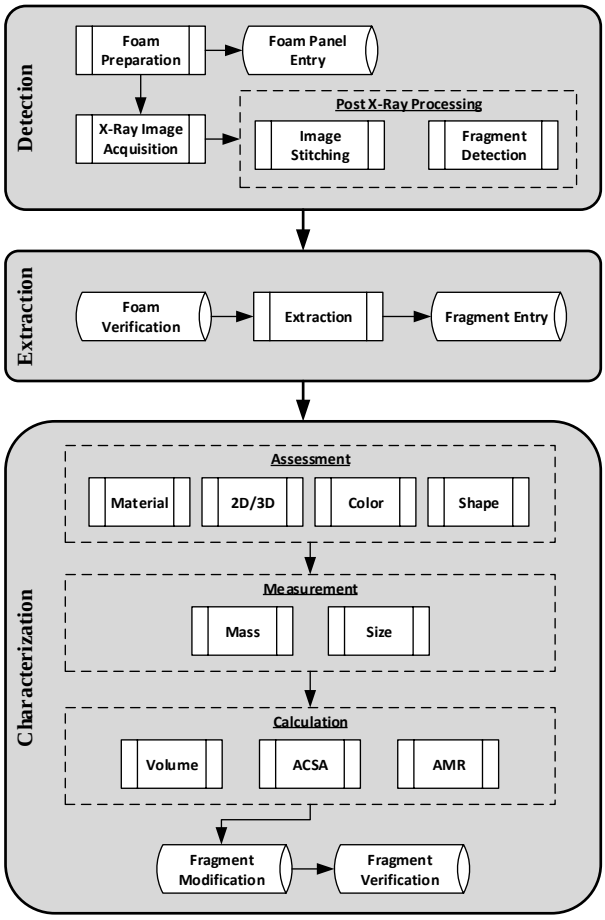

Figure 2: Post-Impact Phase Workflow

Ultimately, the use of a front-end/back-end dichotomy in these types of big data management systems works well when the goal of the application is to maximize efficiency and minimize human error. Further, the design choices made for the back-end services and the principles used to guide the design and development of the front-end interface can be used to guide the design of similar systems for future small satellite missions.

DESIGING FOR SMALLSAT MISSIONS

The front-end – back-end architectural design of the DCS lends itself well to small satellite applications. Typically, small satellite missions will receive raw telemetry and downlink through some sort of back-end layer at a ground station before processing the data to be displayed on some presentation layer. This raw telemetry and downlink can be directly fed into a data management system back-end database engine like the one used on the DCS, then processed and presented on a front-end

interface where users can interact with, modify, or correct recorded data.

The DebriSat DCS uses similar back-end database engine connections such as Open Database Connectivity (ODBC) to feed raw data from external measurement systems into the database engine.[9] These external measurement systems and the use of ODBC connectors are explained in more detail in the next section on the development of the DCS.

Another consideration when designing similar big data management systems for small satellite engines is the selection of back-end services. For the DCS, the Microsoft Windows Server platform was selected as the foundation for the system due to organizational and institutional requirements. Subsequently, Windows-integrated services such as IIS, Task Scheduler, and VSS Backup were selected. Depending on the different application environments for other projects or future small satellite missions, the best option is to select services that are as tightly integrated as possible with the desired operating system platform for the project.

Database engine selection depends on the application and the data set. In the case of the DCS, both the MySQL and the MSSQL database engines were implemented due to the variability and continuous growth in the data set. MySQL works great for development environments and the storage of small data files. MSSQL works great for large production environments and the storage of large data files or filesystems. For small satellite missions, data sets are mostly textual and usually include smaller data files due to bandwidth and access limitations. Therefore, MySQL is likely a better database engine choice for small satellite missions, but of course this highly dependent on the datasets being stored.

Furthermore, data packaging and transferability is a major consideration in data management system architectural design, especially for projects like small satellite missions where data is packaged and moved around frequently. One requirement of the DCS is to facilitate frequent data packaging and transfer between member organizations. As a result, the DCS is designed to store both textual data and data files within the database itself. In other applications where the data set is static, data files are typically stored in a separate location and are referenced within the database as paths.

The alternative storage method employed by the DCS ensures that all data is stored in the same location and eliminates the risk of breaking path links by moving data between environments. This data storage method lends itself well to projects like small satellite missions where data is frequently moved between environments. The DCS storage mechanism is described in further detail in the next section on the development of the DCS.

DEVELOPMENT OF THE DCS

After the design phase of the DCS and the post-impact phase workflow, the development phase began. The DCS and post-impact phase procedures were developed at the same time. As features of the DCS were finalized, their corresponding procedures were implemented physically in the post-impact phase workflow.

Front-End Interface

As in the design of the front-end interface, the development of the front-end interface also paralleled the development of the physical procedures and processes of the post-impact phase workflow. For example, the debris fragment DCS workflow shown in Figure 3 is derived from the "Extraction" and the "Characterization" procedures outlined in Figure 2.

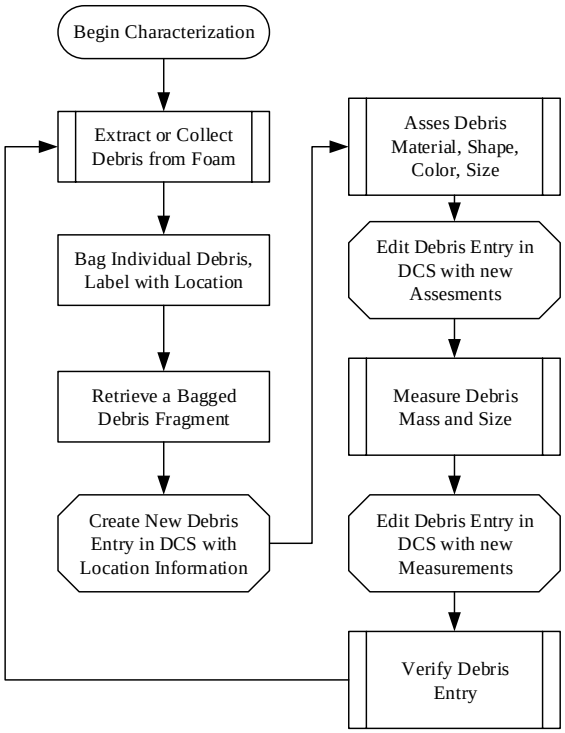

Figure 3: Debris Fragment DCS Workflow

During the development of the DCS front-end interface, procedures like the one shown in Figure 3 guided the layout and organization of the various entry, modification, and verification pages for debris fragments and soft-catch foam panels. For example, the "Add Foam" page is used to create a new debris entry in the DCS. According to the procedures and processes in Figure 2 and Figure 3, only identification and location information must be present when a debris fragment or

soft-catch foam panel is added to the DCS. Therefore, only the identification and location fields are required to submit the "Add Debris" form. Furthermore, the layout of the "Add Debris" page, shown in Figure 4, has been designed and developed to follow the procedure in Figure 3 from the top of the page to the bottom.

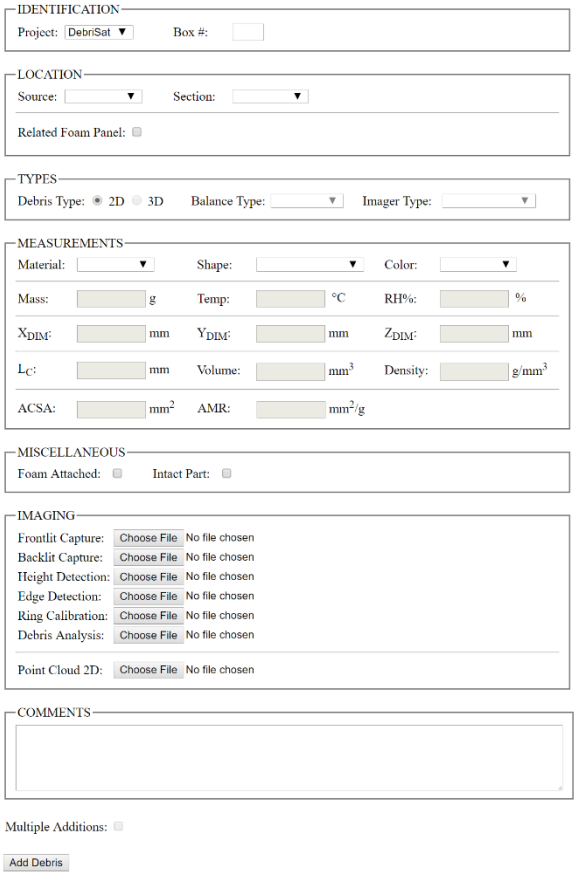

Figure 4: Layout of the DCS "Add Debris" Page

The first two sections of the "Add Debris" page are the first two sections utilized by the user during the initial debris fragment or soft-catch foam panel entry process. The ordering of the rest of the sections correspond to the subsequent phases of the debris fragment assessment and characterization processes. The same layout process applies to the "View Debris", "Edit Debris", and "Verify Debris" pages as well as the corresponding entry, modification, and verification pages for soft-catch foam panels.

Additionally, the forms for each of the front-end interface pages are designed to minimize human error. The number of regular textual input fields is minimized and dropdown selections or checkboxes are used wherever possible to ensure some consistency in user inputs. Furthermore, the forms for each of the front-end interface pages are validated on form submission. For example, the HTML *pattern* attribute is used to restrict the input of the "Box #" field to the format "X-##". If the submitted format differs from the specified pattern, the user is immediately presented with an error message alerting them to the formatting mistake.

The use of these layout and validation design techniques for front-end design helps to maximize user efficiency while minimizing human error. This is critical when the front-end users have direct access to the recorded data.

During the development of the DCS front-end interface, HTML was used to design the layout of the forms (e.g., see Figure 4). JavaScript is used to dynamically change form elements to further guide the user through the post-impact phase workflow. For example, if the user selects the "Source" to be "Embedded" the form dynamically changes to present fields specific to embedded debris fragments. Finally, PHP is used to dynamically fill form fields with data from the back-end database engine. For example, if a related foam panel is available for a debris fragment, a "Foam ID" field is presented using JavaScript and filled with a list of existing soft-catch foam panels using PHP.

Back-End Services

Most of the back-end development on the DCS was done on the database engines. As the post-impact phase procedures evolved and the front-end interface was developed to mirror these procedures, the structure of the back-end database engines was modified to reflect new data fields and data types. The database engines are organized into a tabular structure with separate tables for system activity, general system announcements, user login credentials, dynamic fields, debris data, and soft-catch foam panel data. A description of each of these tables is presented in Table 2.

Table 2: DCS Back-End Database Engine Tables

Table Name	Description
dcs_activity	Log all actions executed, who executed each action, and when each action was executed on recorded data.
dcs_announcements	Store announcements posted by users and administrators of the DCS.
dcs_debris	Store all debris-related data.
dcs_foam	Store all soft-catch foam panel data.
dcs_colors	Store a list of selectable color options for debris fragments.
dcs_materials	Store a list of selectable material options and densities for debris fragments.
dcs_shapes	Store a list of selectable shape options for debris fragments.
dcs_users	Store DCS user login credentials.

Dividing groups of data into separate tables as described in Table 2 help to keep data organized. Further, using tables to store lists of selectable data options enables the use of metadata when querying data. For example, a debris fragment record in "dcs_debris" might have the value "-AL-" stored in its "material" field. Using this information, metadata about the material "-AL-" can be queried to find that it represents aluminum and has an accepted density of 0.002700 g/mm^3.

Each table has its own set of data fields known as columns. Each column has a data type that can range from simple textual data usually stored as VARCHARs to blocks of binary data known as BLOBs.

As new post-impact phase procedures were developed and external measurement systems such as a 2D imaging system were introduced, a storage mechanism for the high-resolution images and data files produced by these systems needed to be developed. Initially, a series of textual columns were added to the "dcs_debris" and "dcs_foam" tables to store paths to these high-resolution images and files, which were stored externally on a server. However, it was clear that if the connection to this server was interrupted or if a file or folder was renamed, moved, or deleted, the paths stored in the database engine would be broken and the association of those images and data files to the original debris fragment record would be lost. The need to package and transfer the data set between members of the project compounded this risk of losing data associations. In response, a new solution was developed where images and files are stored directly in the database engine with the rest of the textual data for each debris fragment and soft-catch foam panel record.

Storing images and files directly in a database engine, like any new feature, has benefits and drawbacks. Storing directly in the database engine dramatically increases the querying capability of the data set. For example, a research can query the back-end database engine to return the raw measurement images for all debris fragments within a specified mass range with a single line of SQL. However, complex queries executed on the entire data set that involve the selection of these images incurs serious performance losses. Thus, careful querying and scripting is required when working with a design in which images and files are stored directly in a database engine to avoid these performance pitfalls.

Ultimately, the need to package the data set frequently and the significance placed on maintaining record associations outweighs the performance impacts of querying the entire data set and the caution required when executing queries. Further, the sizes of the images and data files for debris fragments and soft-catch foam panels are small enough that the query performance when querying a single record is acceptable. More details on the performance of the MySQL database engine when using direct BLOB storage for images and files are presented in the section on the performance of the DCS.

Another major consideration in the development of the DCS back-end layer was the compatibility with external data sources and systems. Recently, three external measurement systems have been introduced into the post-impact phase workflow: a mass measurement system, a 2D imaging measurement system, and a 3D imaging measurement system.[10] These external measurement systems are MATLAB-based and have been designed to use an ODBC connector to interface directly with the DCS back-end layer. An example of the interfacing for the 2D imaging measurement system is shown in Figure 5.

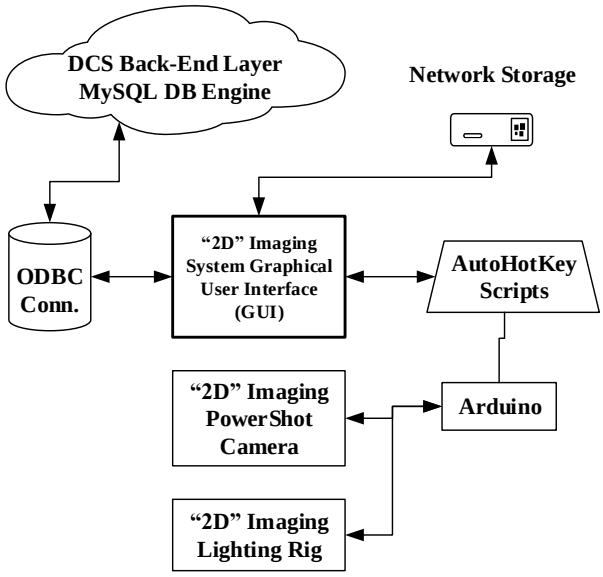

Figure 5: Overview of the Interfacing of the 2D Imaging External Measurement System

The DCS back-end layer was designed to treat external input from these measurement systems like a modification action performed by a user on the front-end interface. That is, a new revision for the associated debris fragment is created, the identification, location, and all other data from the previous revision is filled, and the new measurements from the measurement system are updated in the new revision.

The external measurements systems are used to further minimize human error that may be introduced during measurement or data entry on the front-end interface. Each of the external measurement systems utilize a guided graphical user interface to abstract out the

measurement and calculation process from the user. To use the external measurement systems, a user simply places a debris fragment on the measurement area and the rest of the measurement process is automated. However, the data produced by these measurement systems is formatted and handles by the back-end in the same manner data entered on the front-end interface is. This is important to ensure consistency across the data, maintain record associations, and secure the revision history and chain of custody for each debris fragment.

Future Development

Development continues on the DCS front-end interface and back-end services. As new procedures and data types are introduced into the post-impact phase workflow, the DCS must evolve to support these new processes and fields. The DCS was designed modularly to enable a smooth evolution process that accounts for inevitable changes in the data set structure. For example, the foundational form used on the each of the DCS debris and foam pages can be modified in a single HTML file that applies to all other source files that reference it, use it for data entry, or use it for data display. Similarly, small satellite missions can benefit from similarly robust systems that adapt to changing mission parameters, data types, available downlink and telemetry, etc.

DEVELOPING FOR SMALLSAT MISSIONS

The DCS and the DebriSat post-impact effort have benefitted greatly from the parallel development of the DCS and the post-impact phase workflow. Future small satellite missions with similar big data challenges could equally benefit from designing and developing their ground station data management software in parallel with design and development of the mission. The ability to test various features of the DCS and how they might work with proposed post-impact phase procedures or systems was invaluable in making better design decisions for the project. Similarly, for small satellite missions, being able to test ground-based data management while features in flight software or hardware are being implemented will lead to a more cohesive and synergistic design in the end.

Small satellite missions would likely benefit from first designing the structure of a back-end database engine first. This design should be based on the data types expected from telemetry and downlink. Further, if the spacecraft and data management system are designed synchronously, this structural design can be easily modified to match changes made during the development of the spacecraft's telemetry and downlink formats. Additionally, the interfacing between the back-end layer of the data management system and external data sources such as a ground station TNC or SDR should be decided upon early in the design and development process.

Once a good foundation for the back-end layer has been developed, development should begin on the front-end layer. Likely, future small satellite missions will be concerned more with the display and manipulation of data than with the entry of data. However, the use of form validation and clever design to present inputs and outputs in a logical manner improves the user efficiency when interacting with data and ensures the integrity of the underlying data set.

Developing a robust, modular architecture will enable a smooth evolution and extension process of the data management system in the future. Furthermore, considering the design of the back-end database engine structure early on allows designers and developers to establish good record associations and implement optimization features such as pre-defined list tables like "dcs_materials" or "dcs_colors" shown in Table 2 to leverage metadata to increase the querying power of the data set.

PERFORMANCE OF THE DCS

Finally, the performance of the DCS was evaluated to determine the viability of storing images and files directly in the database engine and to identify the cases where performance may be an issue. Ideally, this performance analysis can be used by future data management systems design to assist in the selection of a database engine and whether or not to store images and files directly in the database engine. This performance analysis was conducted using only the MySQL database engine. In the future, another performance analysis will be conducted with the MSSQL database engine.

To determine the performance of the MySQL implementation of the DCS database engine, the PHPMyAdmin web-based database management interface was used. PHPMyAdmin allows the execution of custom SQL scripts and returns a total execution time for each query. This execution time is the primary metric used to evaluate performance. A screenshot of the PHPMyAdmin interface is shown in Figure 7 and an example of an executed query with total execution time is shown in Figure **6**.

Figure 6**: Example of Query and Execution Time**

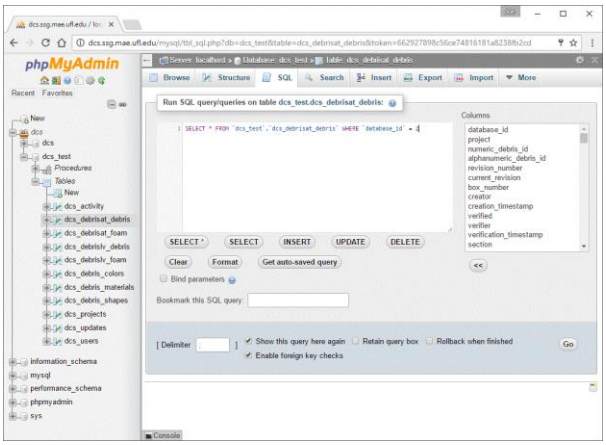

Figure 7: Screencap of PHPMyAdmin Interface

The specifications for the server on which the DCS is hosted and this performance analysis was conducted are presented in Table 3.

Table 3: DCS Server Specifications

Specification	Value
Operating System	Microsoft Windows Server 2016
Processor (CPU)	12-Core Intel Xeon CPU E5-2620 @ 2.10 GHz
Memory (RAM)	32 GB DDR3L 1600 MHz

An example image produced by the 3D imaging measurement system, shown in Figure 8, was used as the test image for uploading to and querying the database engine. The size of the image file is 2 MB, which is the largest image produced when measuring the item shown in the figure.

Figure 8: Test Image from 3D Imaging System

The test image in Figure 8 was uploaded as a BLOB to the database engine using a MATLAB script and the ODBC connector used by the mass and imaging external measurement systems. For the first test, the test image was uploaded to a single image field on the database engine and the total MATLAB script execution time was recorded. For the second test, the test image was uploaded to 126 image fields on the database engine to simulate a full 3D imaging system entry. The results of the first two tests are presented in Table 4.

Table 4: Results of Upload Tests

ID	Test	Execution Time
1	Single Image BLOB	2.09 s
2	126 Image BLOBs	261.67 s

For the third test, a SELECT query was executed on all current DCS data (approximately 120,000 rows) without any image fields populated. For the fourth test, the same query was executed with all 126 image fields populated on a single row of the data set. For the fifth test, the same query was executed with all 126 image fields populated on five rows of the data set. For the sixth test, the same SELECT query was modified to select all fields except the image fields for all DCS rows, with the first five rows of the data set populated with all 126 images. The results of these full database selection tests are presented in Table 5.

Table 5: Results of Full Selection Tests

ID	Test	Execution Time
3	No Image Fields Populated	0.01 s
4	One Row with all 126 Image Fields Populated	14.69 s
5	Five Rows with all 126 Image Fields Populated	N/A – Server Timeout
6	Five Rows with all 126 Image Fields Populated, Only Textual Data Selected	0.09 s

For the seventh test, the first five rows of the DCS data set are selected with no image fields populated. For the eighth test, the first five rows of the DCS data set are selected with all 126 image fields populated. For the ninth test, all the fields except the image fields of the first five rows of the DCS data set are selected, with all 126 image fields populated. The results of these tests are presented in Table 6.

Table 6: Results of Row 1-5 Selection Tests

ID	Test	Execution Time
7	No Image Fields Populated	0.01 s
8	All 126 Image Fields Populated	53.63 s
9	All 126 Image Fields Populated, Only Textual Data Selected	0.01 s

For the tenth test, only the image fields for a single row of the DCS data set with all 126 image fields populated is selected. For the eleventh test, only the image fields for five rows of the DCS data set with all 126 image fields populated are selected. Finally, for the twelfth test, a single image from one row of the DCS data set with all 126 image fields populated is selected. The results of these tests are presented in Table 7.

Table 7: Results from Image Field Selection

ID	Test	Execution Time
10	One Row with all 126 Image Fields Populated	16.41 s
11	Five Rows with all 126 Image Fields Populated	52.64 s
12	One Image from One Row with All 126 Image Fields Populated	0.17 s

The following conclusions can be drawn from the results presented of this performance analysis:

- Querying the database by selecting rows that include image columns populated with image BLOBs causes a large performance impact when the queries are sufficiently large.
- Queries can be written to avoid selecting columns populated with image BLOBs to avoid the performance impacts.
- Most likely, if a large number of images need to be queried, they will need to be dumped through the back-end database engine console.

It is clear that when storing images or files directly in a database engine, queries must be written more carefully to avoid selecting or accessing columns with large BLOB data unnecessarily. Furthermore, the database engine console or some other connecter must be used to dump large amounts of data. The PHPMyAdmin web interface times out if a query takes too long to execute. Therefore, either the timeout of the web interface must be increased or large queries should be executed through a back-end connector.

CONCLUSION

The DebriSat Debris Categorization System (DCS) is well suited for large big data management challenges like the ones posed by the DebriSat experiment and potential future small satellite missions. The DCS is designed with a front-end user interface and a suite of back-end services such as a database engine and backup mechanism. This front-end – back-end dichotomy works well for applications like DebriSat's or future small satellite missions with large data sets where data needs to be stored and accessible via a back-end channel but also presented in a controlled manner to end users. Database engine selection on the back-end layer is a critical decision in the design of a big data management system and is dependent on the desired application. The MySQL database engine works well for development and small-scale applications while the MSSQL database engine is well suited for large, enterprise environments.

The parallel development of the DCS with the DebriSat post-impact phase workflow enabled a high level of synergy between the design of the DCS and the physical characterization procedures implemented for the post-impact effort. Future small satellite missions with similar big data challenges should try to achieve this parallel design and development as much as possible as it produces a more efficient and better suited final product.

Data storage method is another critical decision to make when designing and developing a big data management system. For applications where data is very mobile and must be packaged often, storing small images and files directly within a database engine as BLOBs works well. However, there is a significant performance impact if queries are not constructed carefully.

In conclusion, the DCS serves as an example of how a large and varied data set like DebriSat's can be managed and leveraged to maximize efficiency and minimize error. Ideally, some of the pitfalls and insights discovered during the design and development of the DCS can be used by future projects and, perhaps, small satellite missions to produce similar big data solutions.

ACKNOWLEDGEMENTS

The DebriSat project is funded by the National Aeronautics and Space Administration (NASA) and the United States Air Force/Space and Missile Systems Center (USAF/SMC). The DebriSat team would like to express their sincere gratitude to NASA and the USAF/SMC for their contributions.

REFERENCES

1. J. C. Liou, N. Fitz-Coy, S. Clark, M. Werremeyer, T. Huynh, M. Sorge, M. Voelker, and O. J. Debrisat - a planned laboratory based satellite impact experiment for breakup fragment characterization. In L. Ouwehand, editor, Sixth European Conference on Space Debris. ESA Communications, August 2013a. ISBN 978-92-9221-287-2.
2. Rivero, M., Kleespies, J., Patankar, K., Fitz-Coy, N., Liou, J.-C., Sorge, M., Huynh, T., Opiela, J., Cowardin, H., and Krisko, P., "Characterization of Debris from the DebriSat Hypervelocity Test," Proceedings of the 66th International Astronautical Congress, IAC-15-A6.2.9x30343, Jerusalem, Israel, October 2015.

3. J. Kleespies and N. Fitz-Coy, "Big impacts and big data: Addressing the challenges of managing DebriSat's characterization data," 2016 IEEE Aerospace Conference, Big Sky, MT, 2016, pp. 1-9. doi: 10.1109/AERO.2016.7500889

4. "Home : The Official Microsoft IIS Site", Iis.net, 2017. [Online]. Available: https://www.iis.net/.

5. "MySQL", Mysql.com, 2017. [Online]. Available: https://www.mysql.com/.

6. "SQL Server 2016 | Microsoft", Microsoft SQL Server - US (English), 2017. [Online]. Available: https://www.microsoft.com/en-us/sql-server/sql-server-2016.

7. phpMyAdmin, "phpMyAdmin", phpMyAdmin, 2017. [Online]. Available: https://www.phpmyadmin.net/.

8. "FILESTREAM (SQL Server)", Docs.microsoft.com, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/blob/filestream-sql-server.

9. "Microsoft Open Database Connectivity (ODBC)", Docs.microsoft.com, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc.

10. Moraguez, M., Patankar, K., Fitz-Coy, N., Liou, J.C., Sorge, M., Cowardin, H., Opiela, J. and Krisko, P.H., 2015. An Imaging System for Automated Characteristic Length Measurement of Debrisat Fragments.