Software Requirements Specification

for

Nasa/JPL Parallel SPICE Implementation

Prepared by Matthew Cohen, Nick LaMonica, and Sahib Bajwa

University of Colorado Boulder

3 November 2020

Table of Contents

1. Introduction

1.1 Purpose of Document

This is a Requirements Specification document for a concurrent wrapper implementation for JPL's existing SPICE library. The new implementation will use parallelization techniques to support multi-threaded execution without altering the API experience for the users. This document describes the scope, objectives and goal of the new implementation. In addition to describing non-functional requirements, this document models the functional requirements with use cases. The new library will be called parSPICE.

1.2 Project Summary

Project Name:	JPL Parallel SPICE Implementation
Project Manager:	Austin Albert
Project Developers/Testers:	Nicholas LaMonica Joel Courtney Willie Chew Joseph Ledesma Matthew Cohen Sahib Bajwa
Responsible Users:	Marcel Llopis

1.3 Background

NASA's JPL is a leader in robotic exploration of the solar system. JPL, and other space organizations around the world, utilize a library called SPICE to make geometric calculations in space. The library currently contains wrappers for C, Fortran, Python, Java, and Matlab. While the SPICE library is a very impressive and robust library, it was developed using global variables in a time where multithreaded execution was not a concern. Since then scientists have been using the SPICE library in more creative ways there is a need for a more performant and thread-safe version of SPICE. Marcel Llopis, Modeling and Verification Group Supervisor Planning and Execution Systems Section at JPL, requested that we create a concurrent adapter for the SPICE Java implementation that does not alter the API of the current library.

1.4 Project Scope

The scope of this project includes creating a parallel version for the SPICE Java implementation that will replicate all SPICE functions and support parallel task execution. Aside from the actual system, multiple steps will be put in place to verify the accuracy of our solution. These include result validation tests, performance benchmarks, and a report of project findings. When the parSPICE library is handed off it will come with full documentation and use cases.

1.5 System Purpose

1.5.1 Users

Those who will primarily benefit from the new system and those who will be affected by the new system include:

Scientists:
After the new system is implemented Scientists all over the world that use SPICE to make geometric calculations in space will be able to use the new library to make faster and larger calculations that may have been impossible in the current iteration of SPICE.

Developers:
Developers will be maintaining our code and implementation as time goes on. They will also be adding more functionality to SPICE in the future.

1.5.2 Location

The parSPICE library will be available to the public via NASA/JPL's website.

1.5.3 Responsibilities

The primary responsibilities of the project are:

- Provide a concurrent wrapper for the SPICE Java Implementation.
- Replicate all existing SPICE functions.
- Support concurrent task execution.
- Provide validation tests.
- Provide performance benchmark tests.
- Provide a report of project findings.
- Provide documentation of the new implementation along with use cases.

Other desired features of the new system:

- Provide a wrapper that does not alter the API of the current Java SPICE implementation.

We will not be responsible for documentation of existing SPICE tools, or support for any other SPICE language implementation.

1.5.4 Need

This new implementation is needed to provide scientists with a way of executing calculations in parallel thus allowing them to make more efficient and complex calculations.

2. Functional Objectives

2.1 High Priority

1. The library will serve as a concurrent adapter to Java SPICE.
2. The library will replicate all existing SPICE Functions
3. The library will be able to distribute a set of SPICE calls to sub engines for concurrent execution.
4. The library will also be able to broadcast a command to all sub engines if requested by the user.

2.2 Medium Priority

1. The library will not alter the API of the current Java SPICE implementation

2.3 Low Priority (Stretch Goals)

1. Providing support for an on-prem parSPICE calculation server.

3. Non-Functional Objectives

3.1 Reliability

- The system will make accurate calculations without changing the code of the current JNI SPICE library.

3.2 Usability

- A scientist familiar with the current implementation of Java SPICE should be able to navigate and use the new wrapper with relative ease.

3.3 Performance

- The solution should complete calculation batches significantly faster than the single-threaded execution of native SPICE. Similarly, the performance should scale relative to the number of threads being used for execution. A larger number of active cores should reduce execution time.
- Produce performance comparison benchmarks against native SPICE operations.
- All benchmarks will come with documentation as to what was being tested and what performance was achieved.

3.4 Security

- All requests sent to JPL servers will be authenticated via a handshake.

3.5 Supportability

- The library should be compatible with Linux, Mac OS, and Windows.

3.6 Online user Documentation and Help

- Documentation of the system's architecture.
- Automatically compiled API documentation (javadoc), similar to "vanilla" SPICE.
- Documentation on parSPICE vs "vanilla" SPICE. This includes similarities, differences, and performance gains.
- User Guide that documents best practices and proper use of the library.

3.7 Purchased Components

- Nothing will need to be purchased for this project.

3.8 Altered Requirements

- Any changes to this document will be reviewed and explicitly agreed upon in writing by both parties.

4. The Context and Use Case Model

4.1 Goal Statement

The goal of the system is to provide scientists with a thread-safe SPICE Java implementation allowing them to make more complex and efficient calculations than are currently supported.

4.2 Use Cases

Use Case Name:	Distribute Job (Argument List)
Summary:	The user distributes a set of SPICE calls to sub engines for concurrent execution.
Basic Flow:	1. User creates a command object that maps to the desired SPICE function 2. User adds each required argument tuple to the command object 3. User Initiates the distributed task 4. User waits for results
Preconditions:	1. The user is familiar with the pre-existing “vanilla” SPICE implementation 2. parSPICE must be initialized
Postconditions:	The system successfully distributes the set of SPICE call to n sub engines for concurrent execution.
Business Rules:	The execution of the task will not be altered in terms of user experience in any major way. User should be able to process return values in the same order as their respective arguments

Use Case Name:	Distribute Job (Argument Factory)
Summary:	The user distributes a set of SPICE calls to sub engines for concurrent execution.
Basic Flow:	1. User creates a factory object that can dispense SPICE function argument tuples on-demand 2. User creates a command object that maps to the desired SPICE function and encapsulates the argument factory 3. User Initiates the distributed task 4. User waits for results
Preconditions:	1. The user is familiar with the pre-existing “vanilla” SPICE implementation 2. parSPICE must be initialized
Postconditions:	The system successfully distributes the set of SPICE call to n sub engines for concurrent execution.
Business Rules:	The execution of the task will not be altered in terms of user experience in any major way. User should be able to process return values in the same order as their respective arguments

Use Case Name:	Broadcast Task
Summary:	Upon user request, send a command to all sub engines.
Basic Flow:	1. User creates a command object that maps to the desired SPICE function 2. User adds each required argument tuple to the command object 3. User Initiates the broadcast task 4. User waits for processing completion
Preconditions:	1. The user is familiar with the pre-existing “vanilla” SPICE implementation 2. parSPICE must be initialized
Postconditions:	The system successfully sends a command to all sub engines.
Business Rules:	The execution of the task will not be altered in terms of user experience in any major way.