THE SOFTWARE ARCHITECTURE OF THE UPGRADED ESA DRAMA SOFTWARE SUITE

Christopher Kebschull$^{(1)}$, Sven Flegel$^{(1)}$, Johannes Gelhaus$^{(1)}$, Marek Möckel$^{(1)}$, Vitali Braun$^{(1)}$, Jonas Radtke$^{(1)}$, Carsten Wiedemann$^{(1)}$, Peter Vörsmann$^{(1)}$, Noelia Sánchez-Ortíz$^{(2)}$, and Holger Krag$^{(3)}$

$^{(1)}$*Institute of Aerospace Systems, TU Braunschweig, Hermann-Blenk-Str. 23, 38108, Braunschweig, Germany*
$^{(2)}$*DEIMOS Space S.L.U., Ronda de Poniente 19, 22, Tres Cantos, Madrid, 28760 Spain*
$^{(3)}$*ESA/ESOC (Space Debris Office) , Robert-Bosch-Str. 5, 64293 Darmstadt, Germany*

ABSTRACT

In the beginnings of man's space flight activities there was the belief that space is so big that everybody could use it without any repercussions. However during the last six decades the increasing use of Earth's orbits has lead to a rapid growth in the space debris environment, which has a big influence on current and future space missions. For this reason ESA issued the "Requirements on Space Debris Mitigation for ESA Projects" [1] in 2008, which apply to all ESA missions henceforth. The DRAMA (Debris Risk Assessment and Mitigation Analysis) software suite had been developed to support the planning of space missions to comply with these requirements. During the last year the DRAMA software suite has been upgraded under ESA contract by TUBS and DEIMOS to include additional tools and increase the performance of existing ones. This paper describes the overall software architecture of the ESA DRAMA software suite. Specifically the new graphical user interface, which manages the five main tools ARES (Assessment of Risk Event Statistics), MIDAS (MASTER-based Impact Flux and Damage Assessment Software), OSCAR (Orbital Spacecraft Active Removal), CROC (Cross Section of Complex Bodies) and SARA (Re-entry Survival and Risk Analysis) is being discussed. The advancements are highlighted as well as the challenges that arise from the integration of the five tool interfaces. A framework had been developed at the ILR and was used for MASTER-2009 and PROOF-2009. The Java based GUI framework, enables the cross-platform deployment, and its underlying model-view-presenter (MVP) software pattern, meet strict design requirements necessary to ensure a robust and reliable method of operation in an environment where the GUI is separated from the processing back-end. While the GUI framework evolved with each project, allowing an increasing degree of integration of services like validators for input fields, it has also increased in complexity. The paper will conclude with an outlook on the future development of the GUI framework, where the potential for advancements will be shown.

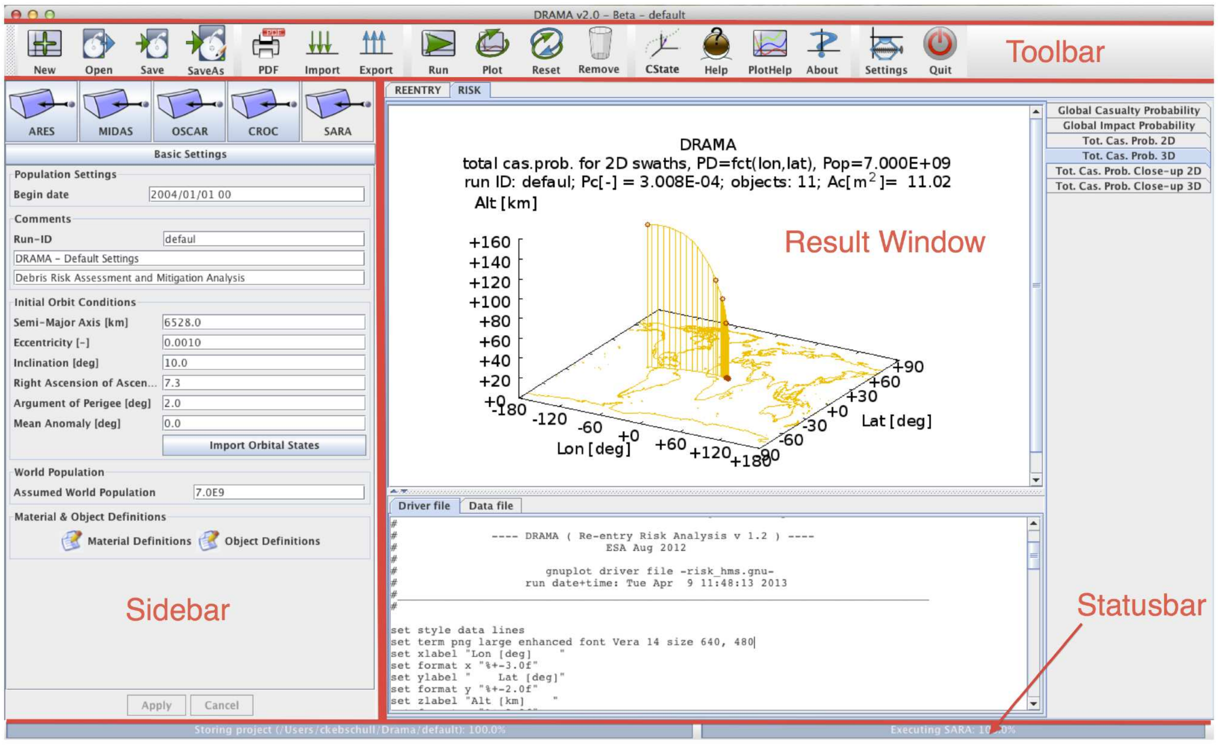
Key words: DRAMA; GUI; architecture; upgrade.

1. INTRODUCTION

Within the upgrade of the DRAMA software suite four of the original tools have been upgraded as well as gained new features. These are ARES (Assessment of Risk Event Statistics), MIDAS (MASTER-based Impact Flux and Damage Assessment Software), OSCAR (Orbital Spacecraft Active Removal) and SARA (Re-entry Survival and Risk Analysis). For the new DRAMA version a fifth tool called CROC (Cross Section of Complex Bodies) has been developed. It is able to calculate the cross section of a complex shaped object. The object model, which the calculation is based on, can be assembled in a 3D user environment developed by DEIMOS [2]. It is part of the DRAMA graphical user interface. ARES as well as MIDAS gained the ability to use MASTER-2009 as a data backend [3]. ARES itself now can use a customizable radar equation as well as new customizable uncertainties and scaling factor look-up tables [4]. In MIDAS the user can specify custom ballistic limit equations (BLE). OSCAR was redesigned so it can model the future solar and geomagnetic activity using a variety of standardized approaches. Also it received an update for investigating de-orbit maneuvers using drag augmentation devices [5]. All disposal scenarios investigated with OSCAR are checked regarding the compliance with the UN Space Debris Mitigation Guidelines [6]. SARA received minor updates for better integration with the new graphical user interface (GUI). While all tools are stand alone command line interface (CLI) applications, they are also embedded in a new GUI, which uses a similar look and feel as MASTER-2009 and PROOF-2009. The layout and the basic features will be described in the following section.

2. FEATURES OF THE GUI

The basic layout of the DRAMA GUI is composed of four areas as shown in Figure 1. The toolbar on the top is responsible to give the user control over the application and the projects. Here new projects can be created,



Population Settings	
Begin date	2004/01/01 00
Comments	
Run-ID	defaul
DRAMA - Default Settings	
Debris Risk Assessment and Mitigation Analysis	
Initial Orbit Conditions	
Semi-Major Axis [km]	6528.0
Eccentricity [-]	0.0010
Inclination [deg]	10.0
Right Ascension of Ascen...	7.3
Argument of Perigee [deg]	2.0
Mean Anomaly [deg]	0.0
World Population	
Assumed World Population	7.0E9

Import Orbital States

Material & Object Definitions: Material Definitions, Object Definitions

```
#
#               ---- DRAMA ( Re-entry Risk Analysis v 1.2 ) ----
#                                ESA Aug 2012
#
#                       gnuplot driver file -risk_hms.gnu-
#                   run date+time: Tue Apr  9 11:48:13 2013
#_______________________________________________________________________
#

set style data lines
set term png large enhanced font Vera 14 size 640, 480
set xlabel "Lon [deg]    "
set format x "%+-3.0f"
set ylabel "    Lat [deg]"
set format y "%+-2.0f"
set zlabel "Alt [km]    "
```

Storing project (/Users/ckebschull/Drama/default): 100.0%

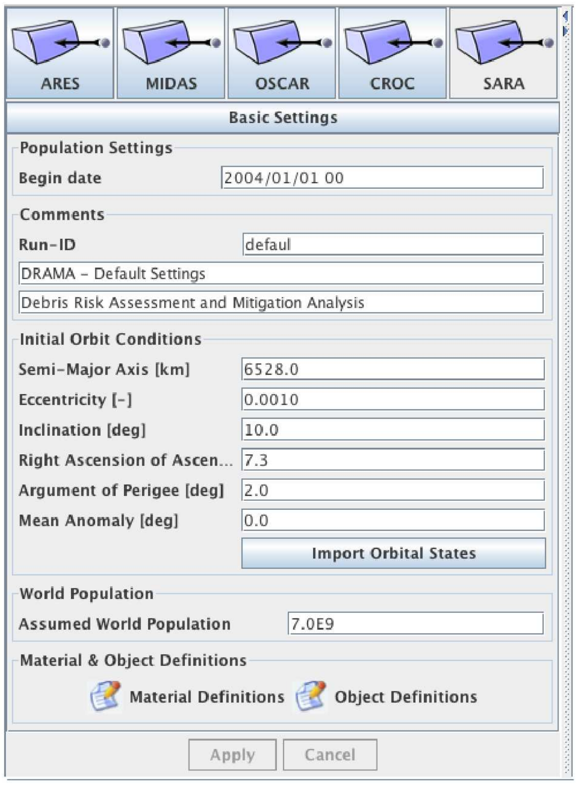
Executing SARA: 100.0%

Figure 1. The layout of the new DRAMA GUI

existing ones can be opened or the settings can be changed. On the left hand side in the sidebar the user can change the settings for each tool that is part of DRAMA. First the tool needs to be selected, which happens on top of the sidebar, as shown in Figure 2. Then one or more settings sections appear below. Using the *Apply* or *Cancel* buttons saves or discards the changes the user has made. Once the settings are in order the user can execute a tool by pushing the *Run* button on the toolbar. The selected tool is being executed and generates its results. The results can be in the form of simple text files or plot driver and data files, which are compiled to image files using the open source tool GNUPLOT. The results are displayed in the result window. Multiple results are grouped into tabs. While the tools are executing, a status window is displayed showing the progress of the computation. The progress is also visible in the statusbar on the bottom of the DRAMA window.

When DRAMA is started up for the first time the user is asked for a workspace directory. The workspace is a folder where DRAMA stores projects. After creating the workspace a first project called *default* is created. The sidebar is filled with settings that are shipped with the software. The user can also create new projects or open existing ones. Each project has an associated folder within the workspace. In the project folder all project related data is stored. When changes to the settings of the tools are made through the sidebar, a tool specific folder is created. It stores the configuration files. Also part of the project handling is the functionality to export projects. When using the corresponding button on the toolbar the current project folder is compressed into a single file, which can be distributed easily. Before executing a tool, the configuration of the settings must be finished. Unsaved changes to the settings are made visible in the sidebar. Using the Apply button in each sidebar section commits these changes. Using the Cancel button these changes are discarded. The user also has the ability to reset all settings either to factory defaults (shipped defaults) or the input files via a button in the toolbar. All entries that are done via the sidebar or an associated dialog are validated through the GUI. Each input field is able to indicate whether an entry is permitted. For example textual inputs are not permitted where decimals are expected. This validation also takes lower and upper limits into account so that inputs can be limited to a specified range. For example the MIDAS particle size range has been set to $[10^{-6}, 100.0]$. The Apply button is greyed out as long as at least one input field in the sidebar section is invalid. This means that the user is not able to commit the changes nor execute the selected tool with the current configuration. Each input field in the sidebar is provided with a tool tip which can be activated by hovering with the mouse over a given field. The content of the tip contains information about the field and upper or lower boundaries monitored by the validation process.

The result window not only displays a given text or image but also related information. For example each image was created by the means of GNUPLOT and



Basic Settings

Population Settings

Begin date	2004/01/01 00

Comments

Run-ID	defaul
DRAMA - Default Settings	
Debris Risk Assessment and Mitigation Analysis	

Initial Orbit Conditions

Semi-Major Axis [km]	6528.0
Eccentricity [-]	0.0010
Inclination [deg]	10.0
Right Ascension of Ascen...	7.3
Argument of Perigee [deg]	2.0
Mean Anomaly [deg]	0.0

World Population

Assumed World Population	7.0E9

Material & Object Definitions

Figure 2. The sidebar showing the content of the Basic Settings section of the SARA tool

therefore also has a linked plot driver and at least one data file. Both are displayed in the lower part of the result window. The user is able to manipulate the plot driver file, store the changes and replot the image within the window. This can be achieved by right clicking on the image or the driver file. All of these files are kept within the project directory in the tool specific output folder. In the case that an image needs to be available as an EPS instead of a PNG file it can be exported by a simple right click on the given image. The dialog that appears when selecting ”Export to EPS” asks the user to specify a folder where the image should be generated. Using the PDF button in the toolbar DRAMA generates a PDF report based on the available data. The user can decide which tool he wants the PDF generated for, as well as which content should be included: results, input files, plot driver files or data files.

The overall settings of the DRAMA application can be changed using the settings menu. Within the settings menu numerous binaries can be specified. For example the user can choose which PDF viewer should be used to show the help files or the generated report. Each tool has its own binary. In the settings the tool specific binary path can be set to a different location. When DRAMA starts up it checks whether the paths to the given binaries are correct and prompts the user for input in case they are not. While all binaries can be executed locally, meaning the machine where DRAMA is started on, in the final version of DRAMA the user can also specify a remote host in the settings, where the binaries will be executed. The remote execution transfers required data to the remote host and retrieves the results once the process has finished. In this use case the local machine does not execute any computations of the tools. It simply displays the GUI and manages the remote executions. This feature however is still under development.

3. ARCHITECTURE

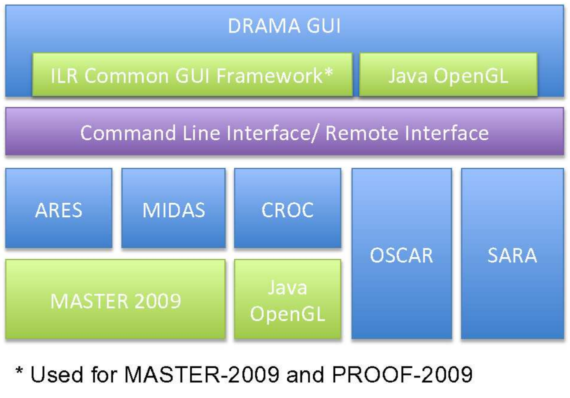

DRAMA GUI
ILR Common GUI Framework*
Java OpenGL
Command Line Interface/ Remote Interface
ARES
MIDAS
CROC
OSCAR
SARA
MASTER 2009
Java OpenGL
* Used for MASTER-2009 and PROOF-2009

Figure 3. The architecture of the DRAMA software suite

The DRAMA software is designed to strictly separate the computations done by the tools ARES, MIDAS, OSCAR, CROC and SARA from the GUI. The tools themselves are command line applications and can be executed independently, as shown in Figure 3. The GUI was developed to be platform independent using the Java programming language. It can be executed on Windows, Linux and OS X given that a Java Runtime Environment (JRE) 1.6 or above is installed. Each tool has been compiled specifically for a given platform. During the installation routine the installer deploys the platform specific binaries of the tools and configures the GUI accordingly. The GUI uses its configuration to locate and call each tool. Furthermore it is able to encapsulate every tools’ configurations into a project structure, which can be modified as needed using the input fields in the sidebar of the GUI, as shown in Figure 4. The GUI monitors the input done by the user and writes it into the respective input files, when all inputs have passed the validation process. After the files have been written, the user can execute the selected tool. The GUI then makes a call to the tool via the command line and sets its working environment to the project directory’s tool specific folder. The tool then is executed within the project and generates its results. Output files are stored in the output folder. An overview of the folder structure is shown in Figure 5. Plot driver files are then passed on to the GNUPLOT executable. It generates PNG image files. These are then displayed in the result window. Results which have been stored as text files are handed over to the result window directly. Each tool

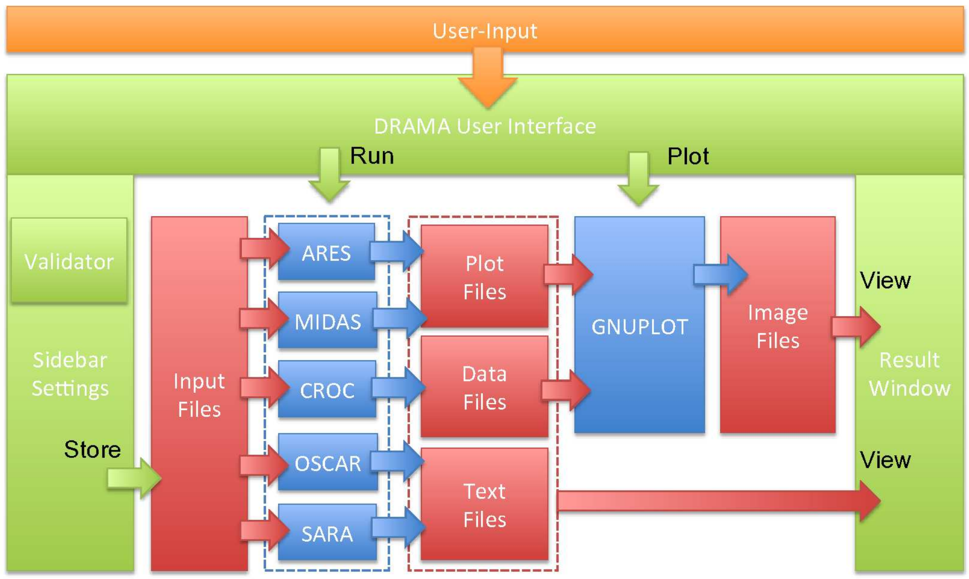


Figure 4. Internal processes of the DRAMA software suite

except SARA supports so called *Run-IDs*. They define which files in a tool's output folder are associated with the current simulation. The generated files with a given Run-ID are removed from the associated output directories, files of different Run-IDs are kept. These results then can be recalled without executing the tool again, given that the correct Run-ID is specified. This can be achieved in the *Basic Settings* sidebar section each tool has.

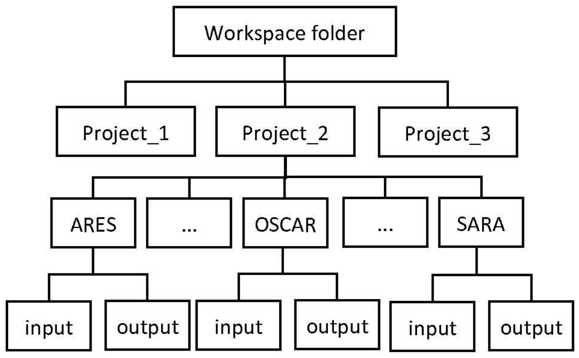


Figure 5. The folder structure of a DRAMA workspace

3.1. Model-view-presenter Paradigm

The GUI was created using the Java language. The software design pattern is based on the model-view-presenter (MVP) paradigm [7]. The basic idea behind the MVP is to separate the data model from the viewing tasks of an application. The presenter part acts as a manager between them, as shown in Figure 6.

- **Model** - holds the information
- **View** - displays the information and receives the user interaction
- **Presenter** - is the link between model and view which manages the synchronization between them and defines the behavior of the application

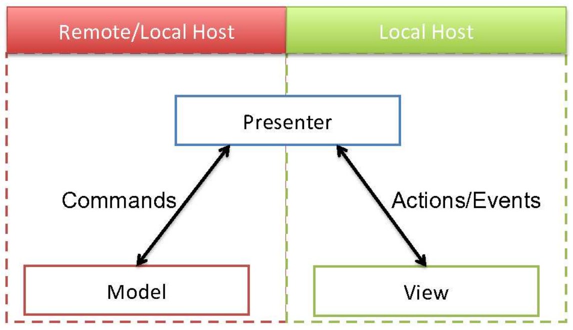


Figure 6. The model-view-presenter design pattern

The GUI relies on the Java Swing library to visualize its views. However by design any other library can be used instead because the view interface definitions do not show any dependencies to a specific graphics library (e.g. Swing, AWT or SWT). In the implementation, however, the graphics library Swing was chosen as the front-end for the application. The chosen MVP approach enables a client-server architecture, where the model can reside on a remote host (server), while the view resides on the local host (client). The presenter is responsible for managing the data model, how it is stored and accessed by the view. Implementing the MVP design pattern results in a more complex project structure and an increased implementation effort. For every input field that is needed in the sidebar three classes have to be created. For a simple text field this results in *TextFieldPresenter*, *TextFieldView* and *SwingTextFieldView* classes in the DRAMA GUI project. The presenter handles how to store the data model, in this case the content of the text field, and acts on the user's input by implementing the validation process. The TextFieldView itself is an interface, that provides the presenter with the methods it needs to control the displayed user interface uncoupled from the implemented graphics library. This includes for example methods for displaying detected input errors by the validator subsystem or enabling and disabling the Apply and Cancel buttons based on the determined validity of the input. The SwingTextFieldView is the implementation of the just described interface using the Swing graphics library. It actually places the input field in the sidebar. It could easily be replaced by a different appearance and graphics layout, while the functionality is guaranteed to remain the same due the interface of the presenter. This defined hierarchy is strictly implemented throughout all inheritance layers of the DRAMA GUI project. Not only makes this approach the front-end replaceable but it also enables the ability for the application to execute the tools remotely (remote data model in the form of input and output files) and view the generated results on the user's desktop computer.

4. FRAMEWORK

The DRAMA GUI was developed with the goal to have the same look and feel as MASTER and PROOF-2009. For this reason a framework had been developed that ensures the same behavior in all future GUI projects. The framework is also based on the MVP paradigm and provides key functionalities, like the basic layout of the main window, as shown in Figure 1 or the ability to manage project structures, read and write input files, execute binaries within the project structure etc. Due to using this framework these core functionalities can be reused in every GUI project and do not have to be redeveloped individually, but can be adapted to fit the needs of the specific project requirements. For example for DRAMA the framework has been extended with new features like handling multiple tools instead of one as in MASTER and PROOF-2009.

Figure 7 shows the parent-child hierarchy as it exists in the DRAMA GUI. The example shows the Basic Settings sidebar section for the SARA tool. From the top presenter layer (ProjectPresenter) to the bottom layer (BeginDatePresenter) all presenter classes and associated view classes are in a parent-child relationship. The BeginDatePresenter is the child to the BasicSettingsPresenter while the BasicSettingsPresenter is a child to the SARAModulePresenter etc. The advantage of building up this kind of hierarchy is first the logical structure itself. In each layer of the hierarchy the parent and child classes are able to interact. The parent class

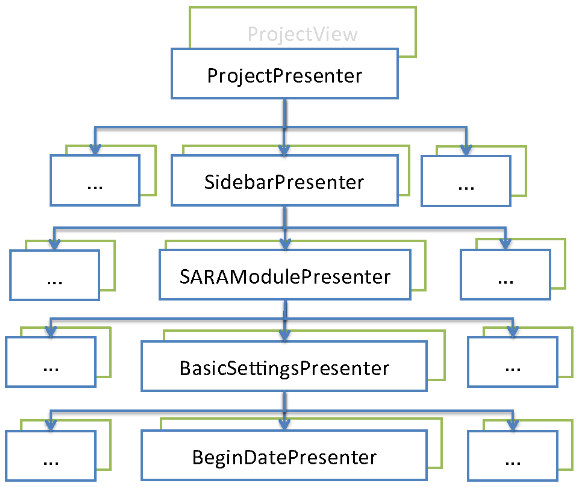


Figure 7. The parent-child hierarchy in the DRAMA GUI. The hierarchy as it exists for the SARA Basic Settings section Begin Date as shown in Figure 2

can broadcast to its children to enable or disable their controls and the child classes can pass up their status. For example the input field can be in a valid or invalid and stored or not stored state. The parent classes can then determine whether the Apply or Cancel buttons should be active.

Figure 8 shows the core functionalities of the framework with respect to their MVP domains. As stated before in the view domain the Java native graphics library Swing has been used to implement the controls the user is interacting with. The interaction can be categorized into *Actions* and *Events*. Actions are direct interactions of the user with the GUI, like pushing a button or typing in an input field. Events on the other hand are triggered reactions of the system to certain parameters like the response of the validator to correct or incorrect inputs in the field. It will cause the Apply button to be greyed out and the warning icon to appear. While Events and Actions are directly connected with the view domain, their processing is done in the presenter domain where an *Event Bus* is available. The Event Bus is built in through all child classes of the *ProjectPresenter*. It enables each layer to act individually on Events. This way dependencies between different sidebar entries can be achieved. Information can be passed to another presenter uncoupled from the parent-child layer they are part of. All presenter instances listen on the same Event Bus. To act on different kinds of Events so called *Event Handlers* are provided by the framework. They can be implemented to act on any Event that is *fired* and broadcasted to all presenters on the Event Bus. With the implementation of an Event Handler for a given Event the presenter is given the means to act on input values. In ARES and CROC for example functionality modes can be selected by the user. Depending on the mode, input fields or entire sidebar entries are enabled or disabled. While enabling and disabling input fields is done in layer four of the parent-child hierarchy the same mechanism for the entire sidebar section has to be applied one layer above. In both layers Event Handlers are implemented, which are able to react on the same Event. In the depicted case it is an *ValueChangedEventHandler* that has to be implemented. The value of the input field will be checked within that Event Handler and the enabled or disabled state will be set accordingly. The value of the input field is part of the model domain. It is however encapsulated within the presenter and monitored by the validator. When the value conflicts with the bounds set for the validator it causes the input field presenter (e. g. the BeginDatePresenter in the sample above) to switch into the invalid state. Due to the parent-child relation the invalid state is passed up to the Basic Settings sidebar section. This behavior of the system also reflects on other parts of the framework like the *Executor Service*. It denies the execution of the SARA binary while at least one configuration value in the corresponding sidebar is invalid. The Executor Service allows for synchronous or asynchronous execution of threads. It queues pending executions and monitors their state. Internally the executables, which are handled by the Executor Service are called Commands. They can have two states. Either they are *dispatched* which means they are in the queue or already executing or they are *released*. When a Command has finished its execution it is released. The state of a Command can also be transmitted through the Event Bus. An Dispatched or Released Event is fired when a Command's state changes. The Command is the last abstraction layer with the presenter domain. It comes directly in contact with the command line or remote interface. Within a Command the operating system depending binary of each tool is called and monitored. Also the Command interface provides a progress monitor as well as a log file reader. As a binary is running, the corresponding Command makes sure that the user is informed of its progress and any available log information is shown in the statusbar. Before a tool is called the *IO Operations* are executed. They write the input files given that the validation process is passed. After a tool has finished executing, its results are read back within the Command and distributed to the result window (output presenters) over the Event Bus. In turn these presenters will either display the results via their associated views or pass them along to the GNUPLOT Command for further processing, as shown in Figure 4.

4.1. Remote Execution

By design Commands can be executed locally or remotely. With default settings the DRAMA GUI is started as a standalone application, which expects its tool binaries (ARES, MIDAS, OSCAR, CROC and SARA) to be available on the local filesystem. With the use of the described Command interface in the framework they are called and executed on the same machine as the GUI. However it is also possible for the Command to be handed over to a remote machine where its execution is realized. For this service to work, a modified version

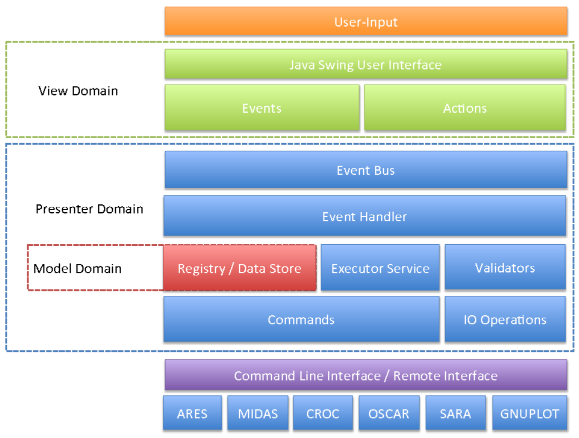


Figure 8. The GUI framework deploying the model-view-presenter paradigm

of the GUI application has to be provided. In this scenario the modified version is the server. It is executed on the remote machine. The Executor Service of both, the client (original DRAMA GUI) and the server are able to communicate by the means of *Java Remote Method Invocation* (RMI) [8]. On the server side all the binaries of the tools have to be present. For the client this is no longer the case. After the server ran a Command, thus executed a binary, it submits the results back to the client where they are displayed or further processed by GNUPLOT. Even the GNUPLOT Command can be executed remotely. This kind of architecture has a couple of advantages for the user. First the user does not need to install the entire DRAMA software suite but rather retrieve the GUI executable, which could simply be done over a web browser. When executing the DRAMA GUI it can be configured so it automatically tries to connect to a dedicated server. The server can be reachable over a companies local area network or the internet. The second advantage for the user is that updates to the binaries and operational data like the solar activity data are managed on the server side. This is especially appealing for unsupported operating systems. While the Java based GUI is executable almost on any platform the tool binaries are restricted to Windows, OS X and Linux on Intel x86 compatible processors. Due to the remote interface, any machine running a compatible JRE is able to execute DRAMA. Please note that the client-server functionality is in the testing stage for MASTER-2009 and is currently under development for DRAMA.

5. FUTURE IMPROVEMENTS

For the DRAMA GUI the original framework, which was used for MASTER-2009 and PROOF-2009, had to be modified so it could handle a more complex setup. The major difference between MASTER/PROOF and DRAMA is the number of tools the GUI is able to control. For MASTER-2009 it was only a single binary that had to be executed. In DRAMA there are five tools and all together six binaries (SARA consists of the binaries RISK and REENTRY). Each binary needs its own input files and generates a number of different results which need to be considered by the result window. During the development of DRAMA this complexity led to an increased implementation effort especially on the IO Operations side. This is due to the fact that the compatibility between the DRAMA file handling and the file handling of the binary has to be guaranteed. With the current process where the GUI generates the input files for the tools and the tools then have to read the files there is an increased chance for errors. Not only are the file handling interfaces written in different programming languages (Java and FORTRAN) but also they are implemented very differently. A lot of testing had to go into this issue to confirm the proper working of this approach on both sides. Another downside to this approach is that at the moment when changes occur in the input file, e. g. the layout is changed by adding a new line to hold a value, both IO implementations have to be altered and re-tested.This overhead of implementing IO handling in both the GUI and the binaries can be reduced by creating a directly callable interface between the Java GUI and FORTRAN binary. In the current process the only interaction between the GUI and the binary is by calling it via the command line interface. The information exchange is done via the input, progress and log files, which have to be written and read by them accordingly. By implementing a new interface a future GUI would not have to read or write any files but rather invoke FORTRAN routines, which are prepared in the binary as shown in Figure 9. The binary itself would handle its own IO Operations while the GUI uses the defined callable routines through the *Management Interface* to exchange information directly. Note that the binary can still be called via the command line interface but there is no command line interface involved between the GUI and the binary. The Java Native Interface (JNI) or Java Native Access (JNA) libraries can be used to achieve these interactions between Java and a binary. The use of this kind of interface would improve the performance of the system because of the drastically reduced use of the filesystem for exchanging information between the GUI and the binaries. The complexity of the GUI project and the implementation and testing effort in the development stage would likely be decreased.

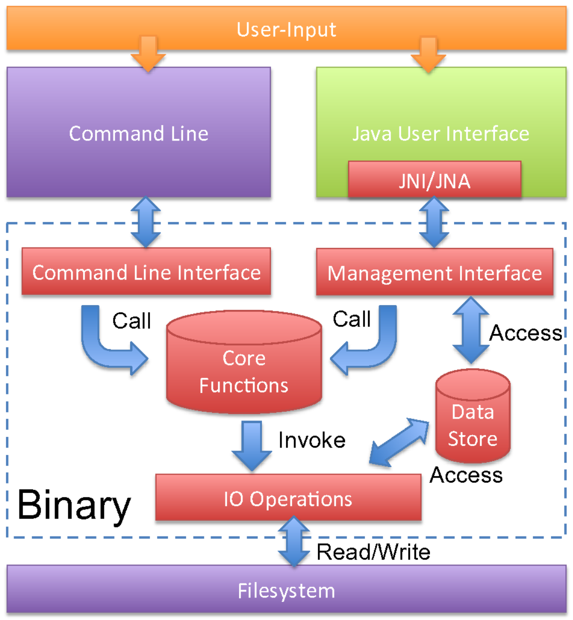


Figure 9. New interfaces enable a simple transfer of information between the GUI and the binary. The filesystem is no longer used for exchanging data.

6. CONCLUSION

A summary of the features of the new DRAMA GUI has been given. Also the general structure and internal processes of the software suite has been discussed. It has been shown that the model-view-presenter paradigm has been used to support the on-going developments for remote execution of the DRAMA tools on a dedicated server by the means of *Remote Method Invocation*. The functionalities of the underlying GUI framework, which had been used in previous projects, has been explained. Due to the increased complexity of the DRAMA GUI, potentials for improvements in future projects could be identified. Utilizing the *Java Native Interface* or *Java Native Access* library could reduce the development effort and increase the performance of the system.

REFERENCES

1. ANON., ESA *Requirements on Space Debris Mitigation for ESA Projects*, 2008.
2. Sánchez-Ortíz N., et al., (2013), *Computation of Cross Section of Complex Bodies in ESA DRAMA tool*, Darmstadt, April 2013
3. Gelhaus J., et al., (2013), *Upgrade of DRAMA - ESA's Space Debris Mitigation Analysis Tool Suite*, 6th European Conference on Space Debris, Darmstadt, April 2013
4. Domínguez-Gonzalez R., et al., (2013), *Update of ESA DRAMA ARES*, 6th European Conference on Space Debris, Darmstadt, April 2013
5. Braun V., et al., (2013), *Upgrade of the ESA DRAMA OSCAR tool*, 6th European Conference on Space Debris, Darmstadt, April 2013
6. ANON., UN *Space Debris Mitigation Guidelines*, 2007.
7. Bower A., McGlashan B. (2000), *Twisting The Triad The evolution of the Dolphin Smalltalk MVP application framework.*, Eighth ESUG Smalltalk Summer School, Southampton, UK, August 28th - September 1st
8. Krüger G. (2002)., *Handbuch der Java-Programmierung*, Addison-Wesley, 3. Auflage.