Preliminary Development of a TSS and SINDA/FLUINT to ESARAD/ESATAN Thermal Model Converter

Kan Yang and Hume Peabody
NASA Goddard Space Flight Center, Greenbelt, MD, 20771

In collaborative spacecraft projects between NASA and ESA, cross-agency exchanges of thermal models are necessary to facilitate thermal design as well as generate temperature and heat predictions at the observatory level. The standard thermal software used within NASA for the Geometric Math Model (GMM) is typically the Thermal Synthesizer System (TSS) or Thermal Desktop, while SINDA/G or SINDA/FLUINT is used for the Thermal Math Model (TMM). However, the standard for the majority of ESA projects remains ESARAD for the GMM and ESATAN for the TMM. Although ESATAN was developed from a SINDA-like framework, several crucial differences in thermal model structure and GMM features make any effort to translate between the two model formats a painstaking task. A recent effort has been initiated by the NASA Goddard Thermal Engineering Branch to develop a standardized converter between the TSS with SINDA/FLUINT and ESATAN/ESARAD formats. Developed in the VB.NET language, this converter tool provides a Graphical User Interface (GUI) to facilitate the conversion process among end users, and employs a framework of two top-level classes for importing, manipulating, and exporting data from GMMs and TMMs. The GMM class stores information regarding input and output files, units, emulation for unsupported surface types, and data structures for Entities, Variables, Optical Properties, and Thermophysical Properties. For surfaces that require emulation, the GMM converter also seeks to devolve more complex geometries into base primitive shapes such that they can be passed between programs. The TMM class stores information regarding nodes, conductors, arrays, and registers, and converts all data types supported by both programs. A preliminary effort has also been made to translate model logic between SINDA/FLUINT and ESATAN. It is hoped that development of this conversion tool will facilitate inter-agency collaborations for NASA and its ESA partners on current and future projects.

I. Introduction

Proposals were submitted to NESC to develop an agency supported utility to address conversion between common ESA and NASA formats for thermal models. Typically, spacecraft thermal modeling consists of two separate but related models: the first is a Geometrical Math Model (GMM), which consists of a collection of surfaces that represent the radiating surfaces of a spacecraft. In addition to their spatial location and size definitions, these surfaces are often further assigned thermo-optical properties, active sides, and node numbers for representation in the Thermal Math Model. The GMM is used to compute View Factors or Interchange Factors for the exchange of radiative energy between surfaces as well as absorbed environmental heating from planetary or solar sources. These couplings and loads are combined with additional electrical heating sources, mass and capacitance effects, and additional linear and radiative couplings to form the Thermal Math Model (TMM). The Thermal Math Model is typically analogous to an electrical network with resistors, sources and capacitances, but where temperatures are solved instead of voltage.

While the formats of the various tools used for GMM and TMMs is different, the underlying approach is quite similar. Therefore, a commonality exists between the tools to allow for conversion between formats, but special attention must be given to the differences. The GMM conversion tends to be more straightforward due to structural limitations in the capabilities of the codes (i.e. surfaces may only be generated within the confines of what the code allows). The TMM conversion is typically more difficult due to the unstructured nature of the logic blocks and the extensibility allowed by the underlying FORTRAN programming language. The proposal was to develop a framework to allow for GMM and TMMs to be imported, manipulated, and exported from a neutral format, initially focusing on GMMs in the ESARAD and TSS formats and TMMs in the SINDA/FLUINT and ESATAN formats. Reader and writer methods were developed along with additional methods for the emulation of features not supported by both codes. The neutral format may also be acted upon to further query the model contents and evaluate or modify them programmatically. The capabilities and limitations of the framework are further described herein.

II. Geometrical Math Model: Framework

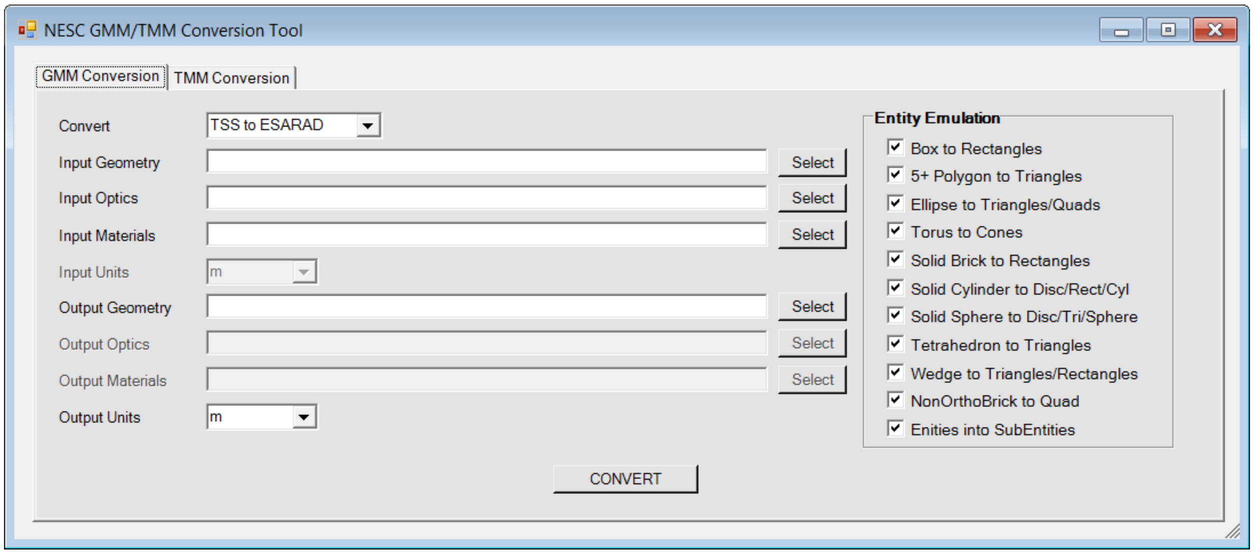
The GMM Converter framework includes the necessary properties to specify the input and output file names, formats and units. It also includes collections of Surfaces, Optical Properties, Thermophysical Properties, Points, and Variables. These collections are populated by dedicated reader methods for the supported thermal analysis codes (currently TSS and ESARAD). Further methods are provided for the emulation of surface types that do not exist in both codes and the representation of subdivided surfaces by assemblies of single node surfaces to account for differences in node numbering order and surface subdivision capabilities of the supported codes. Table 1 shows the list of supported surface types.

Entity Type	TSS	ESARAD	Subdividable?	Emulated?	Emulation
Assembly	Y	Y	N/A	N/A	
Rectangle	Y	Y	Y	Y	Rectangles
Triangle	Y	Y	N	N/A	
Trapezoid	Y	Y	Y	Y	Triangles+Quadrilaterals
Disc	Y	Y	Y	Y	Discs
Cylinder	Y	Y	Y	Y	Cylinders
Sphere	Y	Y	Y	Y	Spheres
Cone	Y	Y	Y	Y	Cones
Paraboloid	Y	Y	Y	Y	Paraboloids
Polygon	Y	N	N	Y	Triangles
Box5Sides	Y	N	N	Y	Rectangles
Box6Sides*	Y	N	N	Y	Rectangles
Torus	Y	N	Y	Y	Cones
Brick**	Y	N	Y	Y	Rectangles
SolidCylinder**	Y	N	Y	Y	Disc+Rect+Cyl
SolidSphere**	Y	N	Y	Y	Sphere+Disc
Tetrahedron**	Y	N	N	Y	Triangles
SolidWedge**	Y	N	N	Y	Triangles+Rectangles
NonUniformBrick**	Y	N	N	Y	Quadrilateral
Ellipse	Y	N	Y	Y	Triangles or Quadrilaterals
Ellipsoid	Y	N	Y	N	
Ogive	Y	N	Y	N	
Hyperboloid	Y	N	Y	N	
Elliptic_Cone	Y	N	Y	N	
Box*	N	Y	Y	Y	Rectangles
Triangle (subdividable)	N	Y	Y	Y	Triangles+Quadrilaterals
Quadrilateral (subdividable)	N	Y	Y	Y	Quadrilaterals
Triangular Prism	N	Y	Y	N	
Half Space	N	Y	N/A	N	
* Box type entities have different numbering methodologies on faces					
** Only outer shell surfaces are emulated					

Table 1. Comparison between ESARAD and TSS capabilities

Writer methods are provided to export the Surface, Optical Property, and Thermophysical Property collections to the specified files for the supported thermal analysis codes (currently TSS and ESARAD). In essence, the framework uses a variant of the TSS formats as a neutral format for the import, storage, manipulation, and export of a thermal GMM. There are currently two major features of codes that are not supported by the converter: use of finite element and edge node numbering and Boolean/cutting operations. ESARAD currently does not support the user defined node numbering of element node numbers to allow for conversion, even though Finite Element support is provided

(ESARAD automatically merges and renumbers the nodes prior to writing the ESATAN file). Therefore, preservation of the Finite Element node numbering from TSS cannot be preserved upon export to ESARAD. This is critical for conversion in order to maintain the link between GMM node assignments and TMM nodal conductor links. Efforts are currently underway to emulate the surface sub-areas associated with a finite element node but using centroid based surfaces. The second major feature not supported during conversion is the use of cutting operations, which may be used in ESARAD. The conversion process preserves the base surfaces needed for the cutting operation and generates them as excluded from radiation calculations in order for the user to be able to re-create representations of the surface in the destination tool.


NESC GMM/TMM Conversion Tool

GMM Conversion | TMM Conversion

Convert	TSS to ESARAD
Input Geometry	
Input Optics	
Input Materials	
Input Units	m
Output Geometry	
Output Optics	
Output Materials	
Output Units	m

Entity Emulation

- ✓ Box to Rectangles
- ✓ 5+ Polygon to Triangles
- ✓ Ellipse to Triangles/Quads
- ✓ Torus to Cones
- ✓ Solid Brick to Rectangles
- ✓ Solid Cylinder to Disc/Rect/Cyl
- ✓ Solid Sphere to Disc/Tri/Sphere
- ✓ Tetrahedron to Triangles
- ✓ Wedge to Triangles/Rectangles
- ✓ NonOrthoBrick to Quad
- ✓ Enities into SubEntities

Figure 1. Graphical User Interface for GMM Conversion

The framework provides a GUI for an end user for the GMM conversion as shown in Figure 1. The framework can also be used by developers to import existing models, further manipulate or evaluate the surfaces, and provide additional capabilities if desired. The source code is written in VB.NET and takes advantage of Microsoft's® .NET Framework 3.5. Therefore, it can be incorporated into any other .NET languages, such as C#, C++, etc for further inheritance or use. The work presented here is currently in the process of being evaluated for release for beta testing, with anticipation of releasing it to industry for general use.

III. Geometrical Math Model: Conversion Examples

Verification of successful conversion was done visually in the source and destination software. Figure 2 shows an original collection of emulatable surface types (Brick, Trapezoid, Box, Solid Cylinder, Solid Sphere, Tetrahedron, Wedge, NonOrthoBrick, Ellipse, and Torus) as well as various incarnations of these entities. Figure 3 shows the converted/emulated shell representation of these entities. The colors correspond to varying optical properties showing the correct application of optical property to the appropriate sides. In some cases, the entities were further subdivided into their respective nodal subdivisions; this is shown by the sub-entities highlighted in yellow to demonstrate their independence from neighboring entities. Upon successful conversion of simple entities, two full observatory models were next utilized for testing: the Lunar Reconnaissance Orbiter (LRO) in TSS and Solar Orbiter in ESARAD. The LRO model was modified to remove the ellipsoid representing the tank and the surfaces with a Finite Element nodalization scheme. Figure 4 shows the original model in TSS on the left and the converted model in ESARAD on the right. Visually, the two models appear nearly identical. A more formal verification will eventually be performed by comparing radiation couplings generated by the two codes and ensure that couplings are within the expected errors associated with the Monte Carlo Ray Trace utilized by both codes.

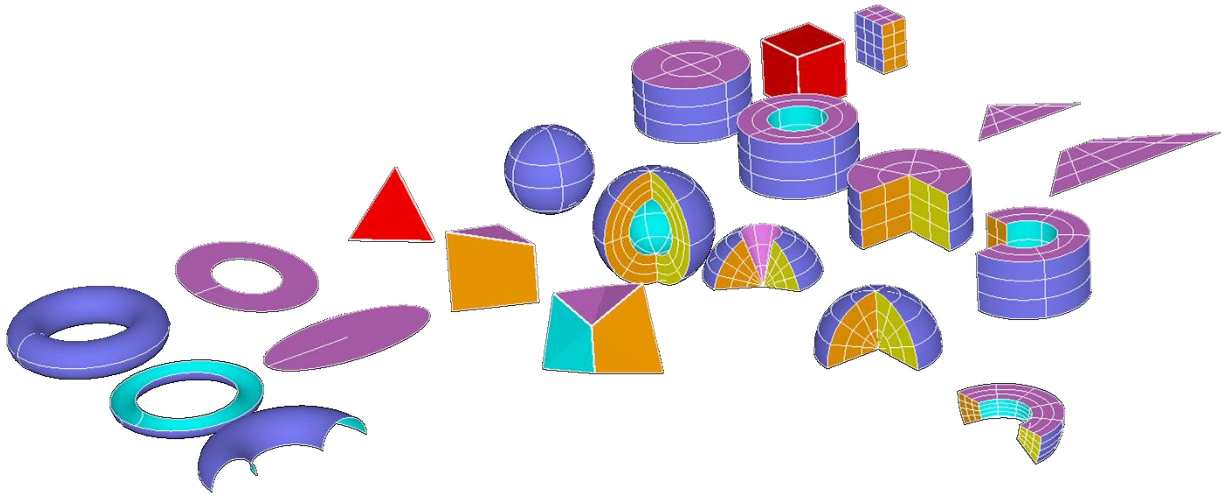


Figure 2. Base set of emulatable entities in native format

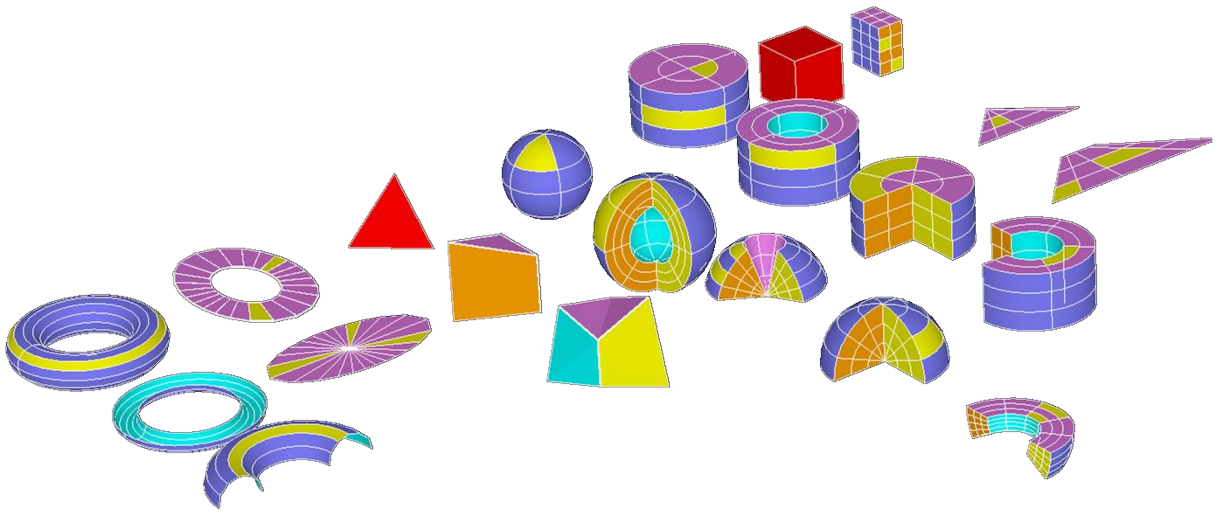


Figure 3. Resultant Emulation Entities after nodal subdivision

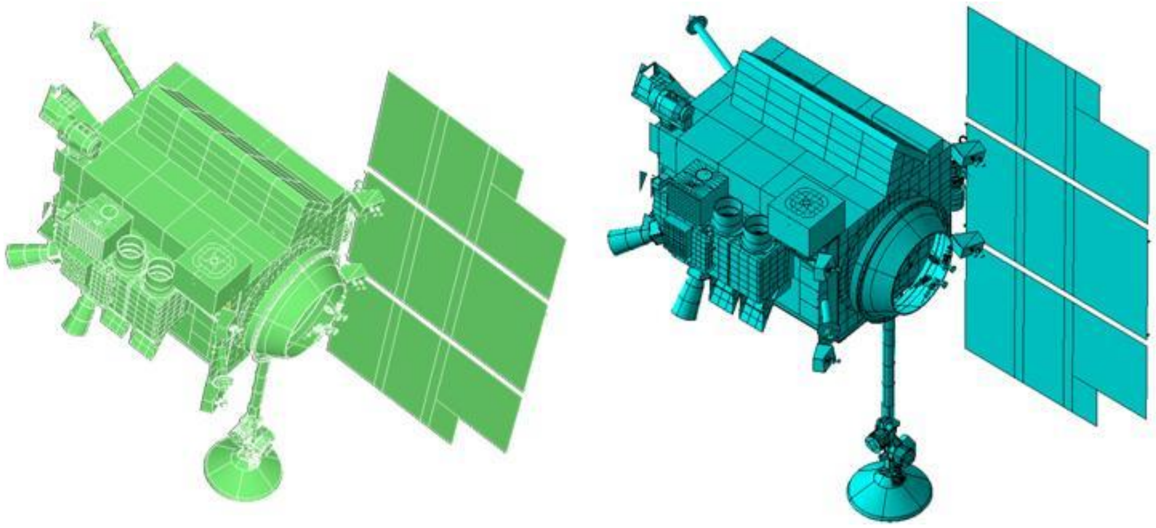


Figure 4. Lunar Reconnaissance Orbiter in TSS (left) and converted to ESARAD (right)

Similarly, the Solar Orbiter model was converted from its native ESARAD to TSS. The two models are shown in Figure 5. It should be noted that the Solar Orbiter model includes numerous cutting operations for penetrations through the Sun Shield, which are not converted. The various blue and yellow surfaces in the left image of Figure 3 represent the base surfaces used in the cutting operations by ESARAD to generate the final geometry seen in the image on the left side of Figure 5.

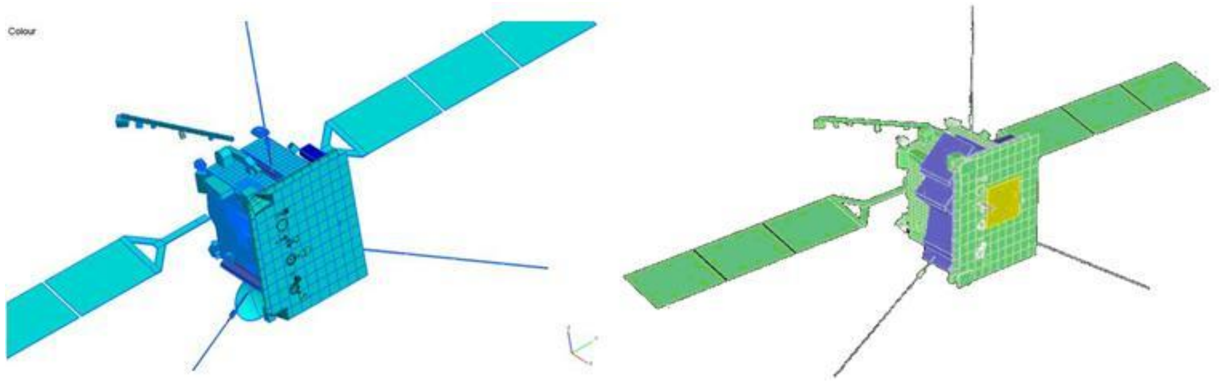


Figure 5. Solar Orbiter in ESARAD (Left) and converted to TSS (Right), cutting operations not converted

IV. Thermal Math Model: Framework

The TMM Converter framework supports conversion between SINDA/FLUINT and ESATAN, the respective TMMs of the TSS/Thermal Desktop and the ESATAN-TMS suites. From user-specified input and output file names, dedicated reader methods are used to populate collections of nodes, arrays, registers, conductors, and comments. From these collections, dedicated writer methods are then used to output into the desired format. While the processing of the data blocks (e.g. Nodes, Conductors, Arrays, etc) is fairly straightforward, the conversion of the logic block is far more challenging due to the unstructured nature of the FORTRAN/MORTRAN language utilized by both codes. Since ESATAN and SINDA each provide their own libraries of function in addition to the intrinsic functions of FORTRAN, a collection of wrapper functions may also be output when converting to replicate the functionality of the origination code in the destination code. These wrapper functions are stored in user modifiable text files such that the source code itself does not need modification to introduce new mappings between function in the codes. However, conversion of the syntax for referencing nodal and conductor properties in logic blocks is provided by the framework.

The ultimate goal of the converter is to create an executable code in the converted format, but due to the wide range of syntax supported by both the SINDA and ESATAN formats, there may be certain sections of code that cannot be converted. Therefore, the converter only attempts to convert a majority of the TMM statements encountered in SINDA and ESATAN codes used by major flight projects, but not every single feature, contingency, or special case. User input will still be required post-conversion, as the code must be checked to ensure an accurate, comparable solution is obtained in the converted format. The following sections will discuss the architectural similarities and differences between the two formats, as well as special considerations taken for the specific SINDA-to-ESATAN and ESATAN-to-SINDA conversions.

Comparison between ESATAN and SINDA Formats

As ESATAN was originally developed based on a SINDA-like framework, it shares many similarities with SINDA. Both programs have data blocks to input variables, nodes, conductors, and arrays in multiple submodels. VARIABLES blocks define time- and temperature-dependent logic to be processed prior to solver execution, as well as allow for the post-processing of logic afterward. In SINDA and ESATAN, HEADER OPERATIONS and $EXECUTION blocks respectively handle the definition of the thermal case and solution parameters, Both programs also allow for user-defined logic in FORTRAN or MORTRAN which have the additional capability of referencing properties defined in the data blocks. These similarities and differences are shown inTable 1. Comparison between ESARAD and TSS capabilities

BLOCK	Description	ESATAN	SINDA
NODE	Diffusion Node	Y	Y
	Arithmetic Node	N (Diffusion with C=0)	Y
	Boundary Node	Y	Y
	Inactive Node	Y	N
	Heater Node	N (Same as Boundary)	Y
	Nested Submodel Hierarchy	Y	N
	Global Submodel References	N	Y
	Node Variables Created	T,QS,QA,QE,QI,QR,A,EPS, ALP,L,C	T,Q,C
	Simultaneous creation of multiple nodes	DO Loop	Supported: GEN, SIM, DIM statements
	Nodes with temp-dependent capacitances	INTERP function call in C value	Supported: SIV, DIV statements
SOURCE	Heat Source Reference	N/A	Node Number
COND-UCTOR	Linear Conductor	Y	Y
	Radiative Conductor	Y	Y
	Conductor Reference	Node Pair	Number
	Simultaneous creation of multiple conductors	DO Loop	Supported: GEN, SIM, DIM statements
	Conductors with temp-dependent conductances	INTERP function call in conductor value definition	Supported: SIV, DIV statements
	Trans-submodel connections	Only at higher submodel	Global
ARRAY	Singlet and Doublet Arrays	Y	Y
	Bivariate and Trivariate Arrays	Supported in $TABLES	Y
	Array Reference	String	Number
CARRAY	Character Array Support	N	Y
VARIABLES	Global Variables Submodel Specific Variables Submodel Specific Variable Reference	N Y String	Y Y Number (K,XK)
	Submodel Specific Variables Globally Accessible	N	Y
LOGIC	Model FORTRAN Entry Point	Y	Y
	Include Additional Files	$INCLUDE	INCLUDE, INSERT
	Initialize Values	Y	N
	Node and Conductor Variable scope	Only at higher submodel	Global
	Q values reset to SOURCE DATA or zero at start of Timestep	N	Y
	Instructions for start of Timestep (VARIABLES 0)	N	Y
	Instructions for start of Iterations (VARIABLES 1)	Y	Y
	Instructions for Post Convergence (VARIABLES 2)	Y	Y
	Instructions at regular Ouptut Intervals	Y	Y
	User Subroutines	Y	Y

Table 2. Comparison between ESATAN and SINDA capabilities

As seen in Table 2. Comparison between ESATAN and SINDA capabilities SINDA and ESATAN have many inherent architectural differences. One of the most prominent differences between the two formats is that the ESATAN model structure allows for a hierarchy of submodels where one submodel may be nested in another as a child submodel. In this construct, each submodel may be separately analyzed as a standalone model, and the nodes, conductors, and variables defined in a child submodel can only be accessed by the parent submodel. However, the child submodel can only access local information, not any definitions in its parent or sibling submodels. SINDA, in contrast, places all of its submodels on the same global level and any submodel may reference entities in another.

In the Node and Conductor data block definitions, the most common format includes only the basic node and conductor information for single creation of simple values. SINDA includes a number as the unique identifier for a conductor whereas ESATAN utilizes the node pair as the unique identifier. For the case of multiple conductors in ESATAN between the same node pair, an additional index is implicitly included in the definition. SINDA features special constructs for the generation of multiple or varying values: SIV and DIV statements allow for array-based nodal capacitance or temperature-varying conductance, and GEN and SIM statements create multiple instances of simple value inputs and SIV statements. ESATAN does not allow for generation of a group of nodes or conductors in one statement, but it may be accomplished with a DO loop structure. Furthermore, ESATAN allows the use of library functions in the capacitance and conductance definitions; SINDA only allows intrinsic FORTRAN statements (e.g. EXP, SIN, etc.). Therefore, SIV conductors or nodes in SINDA could be emulated with calls to the INTERP function in the ESATAN library.

SINDA assigns numbers to identify arrays and allows for the use of singlet, doublet, bivariate, and trivariate inputs, whereas ESATAN defines its array names with strings and populates its arrays with the type of variable as defined under the current block declaration (e.g. $INTEGER or $REAL). ESATAN allows the definition of singlet and doublet arrays within its $ARRAYS block and provides explicit array or matrix size definitions after the array name. For bivariate or trivariate arrays, ESATAN defines these in a separate $TABLES block. MIXARRAY formats, where both integers and floats are defined as array entities, as well as character arrays are not convertible from SINDA due to their lack of support in ESATAN.

SINDA execution begins in the HEADER OPERATIONS block, whereas ESATAN execution begins in the $EXECUTION block. ESATAN has an added feature of an $INITIAL block, which allows initialization of data prior to execution, effectively acting as the first "subroutine", as the program executes the $INITIAL block code only once. Logic in the $INITIAL block of ESATAN is replicated at the beginning of OPERATIONS in SINDA. In SINDA, Variables 0 handles the time-dependent logic and Variables 1 the temperature dependent logic prior to solution of the heat transfer equations; Variables 2 handles the post-processing. In ESATAN, only the Variables 1 and 2 blocks exist as an equivalent analogy to SINDA, with both time and temperature dependent logic being in the $VARIABLES 1 block. Figure 6 and Figure 7 graphically show the comparison of SINDA and ESATAN operations flow for Steady-State and Transient cases, respectively.

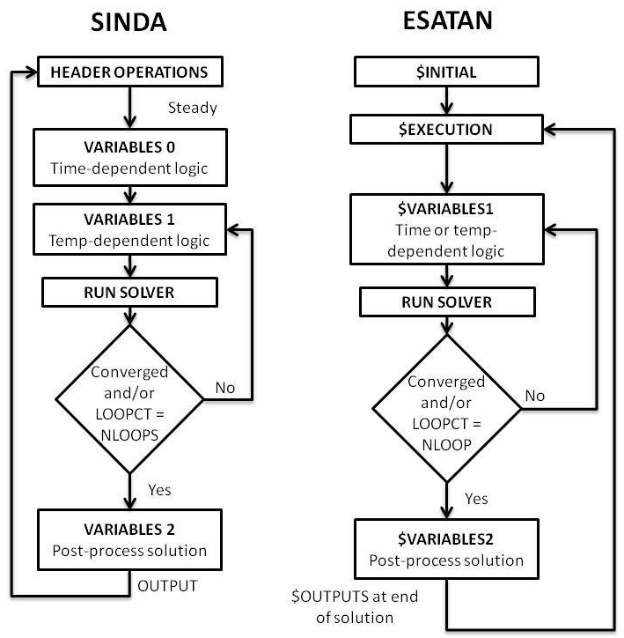


Figure 6. SINDA and ESATAN Operations Flow for Steady-State Cases

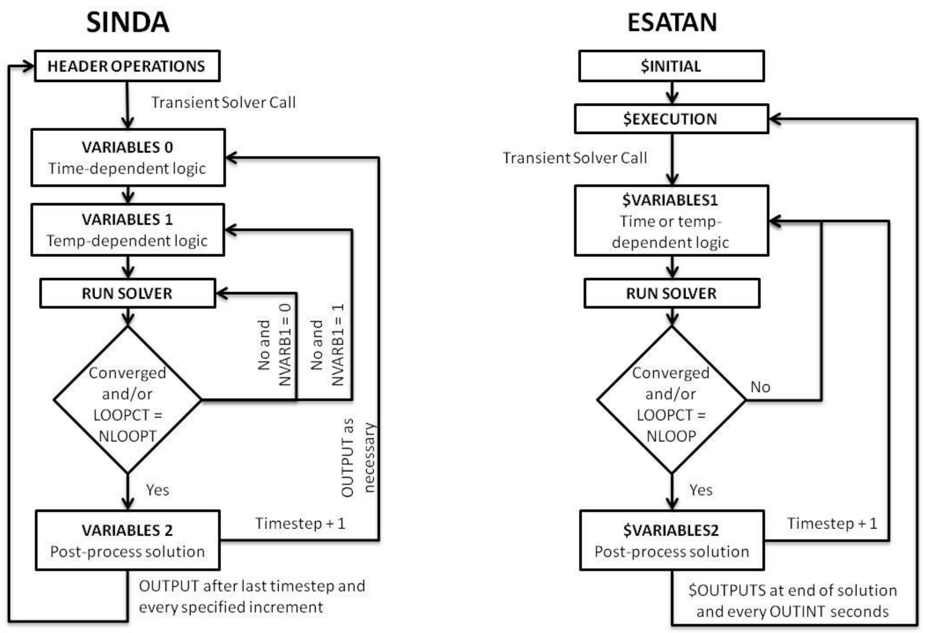


Figure 7. SINDA and ESATAN Operations Flow for Transient Cases

Conversion from ESATAN to SINDA

Due to the intrinsic differences between two formats, certain features must be removed or emulated from each format to allow them to be converted. For conversion to SINDA, the ESATAN submodel hierarchy must be removed so that all of the submodels are on the same level. However, this prevents duplicate submodel names or duplicate variable names under different submodels from being used. Where possible, the current work attempts to convert equivalent constructs in both formats.

In the nodes and conductors block, ESATAN allows for an array-based variation of capacitance or conductance in its node and conductor definitions, respectively, via an INTERP function call. When converted to SINDA, these array-dependent nodes and conductors are assigned an initial, fixed value in the data blocks and then updated with array interpolations in the VARIABLES 1 block. SINDA and ESATAN have directly translatable equivalents for node type in diffusion and boundary nodes; however, in ESATAN, arithmetic nodes are created as a diffusion node with capacitance equal to 0. When converted to SINDA, they are created as diffusion nodes with a fixed value; the ARITNOD subroutine is then called in VARIABLES 1 to re-cast it as an arithmetic node for the solution. This precaution is used such that if the node was defined initially in ESATAN with no capacitance, and a capacitance was later specified in logic, the SINDA compiler does not fail on a node with a capacitance of 0.

ESATAN has an additional feature of an "inactive" node. Inactive nodes are ignored during the solution routine, but their properties may be referenced at any other point within the code. For conversion to SINDA, inactive nodes are re-cast as heater nodes using the HTRNOD subroutine in SINDA, and all subsequent references to the node in the CONDUCTORS block are commented. Thus, the properties for the node are able to be referenced in SINDA without the node having any impact on the resultant solution. ESATAN also allows for "optional" properties to be defined in the nodal input statement, such as heat from environmental and internal sources, area, absorptivity, emissivity, and location in Cartesian coordinates. Since SINDA does not accommodate these, all untranslatable properties are placed into the inline comment as reference, with the exception of heats, which are added later in the VARIABLES 1 block. For references to any nodal properties in ESATAN for which there are no SINDA variable equivalent, such as areas (which are often referenced in the radiative conductance terms for MLI couplings in ESATAN) these variable references are replaced in SINDA with their associated values.

In ESATAN, conductors may only reference nodes in that submodel or its children. Conductors cannot reference nodes in sibling or parent submodels. This is very different from SINDA, where all nodes can be referenced in any conductor block with the proper submodel prefix. Hence, to avoid the unnecessary task of tracking submodel hierarchy to reference conductors, the converter automatically places all conductors in a global submodel and removes the submodel hierarchy in ESATAN, thus making it consistent with SINDA in which all submodels are at the same level. The converter also explicitly places the submodel name before the node number for all conductor generation statements, removing any ambiguity as to which submodel the node number references. Conductor numbers are used by SINDA to refer to the pair of nodes for which the coupling is generated in the conductor statement. ESATAN just directly references the node numbers themselves. For example, in SINDA, a conductor definition may be written as:

```
5, MAIN.2, BATTERY.3, 15.0
```

Where the first entry is the conductor number, the second and third are the nodes for which the conductors are being generated, and the last is the conductance value. The same statement in ESATAN may be written as below, but only from the parent submodel of both MAIN and BATTERY:

```
GL (MAIN:2, BATTERY:3) = 15.0;
```

If there is a reference in logic for a particular conductor, SINDA references the conductor number directly (e.g. G5) whereas ESATAN references the entire node pair (e.g. GL(MAIN:2, BATTERY:3)). After conversion, all ESATAN conductors are assigned sequential conductor numbers, keeping the linear conductors and radiative conductors in separate ranges. References to conductors in logic are translated to the appropriate conductor number reference.

Arrays were similarly handled as conductors with ESATAN array names being assigned sequential array numbers, with the pre-defined array name in ESATAN placed in the SINDA inline comment. A special provision is made for the ESATAN "shorthand" array input, in which the "@" symbol could denote many subsequent instances of the same value. For example, "5@1.0" denotes that the array should be populated by the value "1.0" five times. This is fully expanded in the array input before storing into the array collection. References in logic to the array name are replaced with the SINDA array number reference.

The unstructured nature of the logic statements complicates conversion of ESATAN to SINDA logic. Wherever possible, wrapper functions were provided in a separate subroutine library to replicate ESATAN subroutines with SINDA subroutines. For example, if an array interpolation routine such as INTRP1 in ESATAN is used, after conversion the original function call is preserved to call INTRP1 in SINDA. Since no such subroutine exists in SINDA, in the subroutine library a separate wrapper function is written with the name INTRP1 that calls upon intrinsic SINDA functions (such as D1DEG1) to complete the array interpolation. With this method, though, user routines supported via wrappers may not convert correctly. References to ESATAN node variables (e.g. T, QI, etc) are replaced with their SINDA equivalents.

Conversion from SINDA to ESATAN

The reverse conversion from SINDA to ESATAN provided a set of challenges separate from the ESATAN to SINDA conversion. The hierarchical structure of ESATAN submodels does not exist within SINDA, and therefore all conductors, arrays, variables, and logic are moved to the top level in the converted ESATAN format to allow global access. This includes SINDA logic previously written for a specific submodel. In this manner, only node data and numbered user data is defined within a submodel and all conductors and logic are placed into the top level parent submodel.

In the data blocks, since ESATAN does not support generation of multiple nodes or conductors in one statement, all GEN and SIM statements in SINDA are expanded before storing in memory. For nodes and conductors based on expressions or SIV/DIV statements, these are assigned a fixed value in the data blocks and then later computed in the VARIABLES blocks. Also, due to the difference in how ESATAN references conductors and arrays, all SINDA conductor numbers were removed and stored in the inline comment. Since ESATAN defines its array names with strings, a prefix of “ARRAY” was placed before SINDA array numbers such that they would be clearly defined in the ESATAN code.

Recognition of bivariate and trivariate arrays were difficult in SINDA and therefore the converter only takes a rudimentary effort at identifying the type of array based on the number format (integer or float) and sequence of each array. Also, definition of array lengths in SINDA is difficult since SINDA does not explicitly specify the ends of arrays, and therefore other reference markers (such as the beginnings of other arrays or end-of-file statements) are used to extract array length. Although ESATAN is more explicit in its definition of bivariates and trivariates with its $TABLES block and array headers, conversion of arrays beyond singlets and doublets still proves a challenging task.

To reference properties of nodes in ESATAN logic, the SINDA syntax for this reference is converted to ESATAN syntax (e.g. from “SUBMODEL.T56” in SINDA to SUBMODEL:T56 in ESATAN). Since ESATAN does not use conductor numbers, the corresponding node pair must be found for the assigned conductor number in SINDA and then referenced in the logic whenever there is a call of a conductor property. Furthermore, any K or XK variables created in a particular submodel’s user data block in SINDA must be converted to the equivalent ESATAN submodels under the appropriate $INTEGER and $REAL blocks. It should be noted that SINDA uses the EQUIVALENCE capability of FORTRAN to allow two variables to be stored at the same address in memory. Therefore, referencing XK would process the bits as a REAL, while K would process the same bits as an integer. This equivalencing is not preserved on conversion and separate variables (K and XK) are created for each number.

In logic, SINDA has certain HEADER blocks that are not immediately convertible to ESATAN. For the HEADER SOURCE DATA block, the information in SOURCE DATA is transferred to the $VARIABLES1 block in ESATAN where it is added as impressed heat (QI) to its destination node. VARIABLES 0 logic is also transferred to the $VARIABLES1 block since VARIABLES 0 has no equivalent in ESATAN. In the model execution block, keywords that are used to call a solution routine are replaced if they have an equivalent routine in the format they are being converted to. Control constants and intrinsic conversion constants are also replaced if they have an equivalent constant name in the output format. For other variables, if they exist in the registers block, they are directly converted to the output format without modification.

In SINDA, nodal Q variables are reset to zero at the start of VARIABLES 0. However, in ESATAN, the previous value from the last timestep or iteration persists upon entering VARIABLES 1. Therefore if a SINDA model included all additive instructions (Q1 = Q1 + 1.0) counting on the reset, ESATAN would cumulatively increase the value while it would be constant for SINDA. Therefore, all nodal Q variables are reset to zero using the SETNDR function in ESATAN. For all other intrinsic functions and subroutines, wrapper functions are used for conversion between formats, synonymous with those mentioned previously for the ESATAN-to-SINDA conversion.

V. Conclusions and Future Work

A TSS/SINDA to ESARAD/ESATAN conversion tool was developed by the Thermal Branch of NASA Goddard Space Flight Center. The GMM converter provides a method for converting radiation surfaces between TSS and ESARAD. Additional functionalities are provided for the emulation of surface types that do not exist in both codes as well as the representation of subdivided surfaces by assemblies of surfaces. Previous thermal model examples were used to test the conversion capabilities of the ESARAD converter. In addition, a TMM converter was developed to convert between SINDA and ESATAN. Data block conversions were largely supported due to their structured nature, but only general functions and references were supported for the logic blocks. Special considerations were made specific to the ESATAN-to-SINDA and SINDA-to-ESATAN conversions due to the intrinsic differences of each code.

Due to the extensive variations in syntax as well as intrinsic functions supported by both formats, there remain many opportunities for future expansion of the converter. As examples, the code could be adapted to support the SPV and DPV node and conductor declaration formats in SINDA, which allow for polynomial-based variations in nodal capacitance and conductance. A more robust bivariate and trivariate array detector could be included, or a more robust subroutine library which contains a greater number of wrapper functions for conversion between ESATAN and SINDA. Since the subroutine library is also provided as a text file, the user is able to add additional wrapper functions applicable to specific model conversions. Through the current work, it is hoped that the converter provides a robust basis for TSS/SINDA and ESARAD/ESATAN conversion with an easily expandable architecture for future additions that facilitate inter-agency collaborations.