



HAL
open science

The Sissy Electro-thermal Simulation System - Based on Modern Software Technologies

G. Horvath, A. Poppe

► **To cite this version:**

G. Horvath, A. Poppe. The Sissy Electro-thermal Simulation System - Based on Modern Software Technologies. THERMINIC 2005, Sep 2005, Belgirate, Lago Maggiore, Italy. pp.51-54. hal-00189452

HAL Id: hal-00189452

<https://hal.science/hal-00189452>

Submitted on 21 Nov 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



THE SISSY ELECTRO-THERMAL SIMULATION SYSTEM – BASED ON MODERN SOFTWARE TECHNOLOGIES

György Horváth, András Poppe

Budapest University of Technology and Economics,
Department of Electron Devices
gyuri@eet.bme.hu, poppe@eet.bme.hu

ABSTRACT

Due to the increasing component density and operational speed of today's integrated circuits the dissipated power and the chip temperature increases as well. Designers have to consider the thermal effect of the layout already in the conceptual design, when usually the circuit schematic is available only. This paper presents an easy-to-maintain simulation system which provides electro-thermal simulation already in the conceptual design either on chip or on board level.

The XML descriptions are read by the **xJ** system at runtime. Data structure segments are created on the fly from the templates defined in the XML files. Extending the data structure, for example add new parameters to a component, is simple and fast because only the definition XML files have to be edited, recompilation is not necessary. Since no program code is generated by this technology during the development the later refinement of the program architecture is easy and needs minimum extra coding time.

1. INTRODUCTION

In the early 1970-ies at the Department of Electron Devices of BUTE the TRANS-TRAN circuit simulator has been completed with the treatment of thermal effects [1] resulting in a solution method known as *simultaneous iteration* [2]. This type of self-consistent electro-thermal simulation needs a model of the thermal side that can be treated by the circuit simulator. In our program package this thermal model is generated by the thermal characterization of the chip (PCB) layout, performed by a thermal simulator [3]. The recent system is called *Sissy* (previous implementations were called *SISSI*). In electro-thermal simulations different programs (solvers) need to be used for the same objects. The components have electrical models and they also have a layout representation. This heterogeneous description of the same object however, has to be treated in the simulation environment in a homogeneous way. To resolve this contradiction we developed a new technology: separating data structures from algorithms and from user interfaces (see Figure 1). For that purpose an appropriate extension of the JAVA language is used: we call the system **xJ** and put classes into a new package called **xj**. This way we realized a flexible, easy-to-maintain simulation package that supports different types of electro-thermal simulation tasks either on chip or on board level.

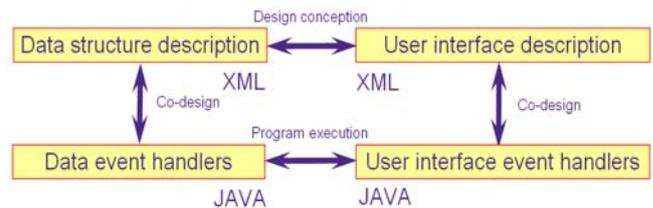


Figure 1: Major components of the software model used in Sissy

The connection between XML data definitions and JAVA classes is established through class names. Class hierarchy is defined in the program code, data type hierarchy follows this through constructors.

The following figure illustrates the data structure creation from template sub-structures.

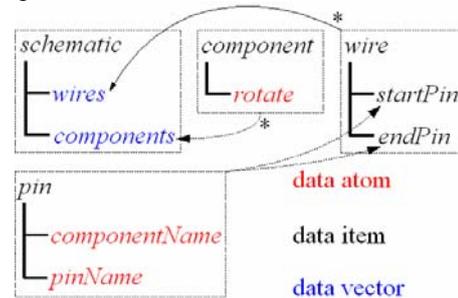


Figure 2: Data structure hierarchy in Sissy

2. THE TECHNOLOGY

As shown in Figure 1 the data structure and user interface can be designed separately by the appropriate tools. Only the necessary event handler code segments should be written by hand.

Data structure elements have the following hierarchy

- *Data atoms* are leaves in the data tree. Each of them has one unique class that manages it when the program

runs. This class implements the `xj.DataAtom` interface.

- *Data items* are aggregated types: built up from data atoms and data items. They have name and description attributes. They are managed by subclasses of `xj.DataItem`.
- *Data vectors* are data items with the possibility to connect several data item instances to one data vector instance. This is the way of instancing. They are managed by subclasses of `xj.DataVector`.

In a program using **xJ** there exists always a root data vector. All data items can be accessed from this root node. We use path expressions similar to file systems.

The `/` path refers to the root data vector. New data item instances are connected to this node by default. To access a child in a vector we use the child's name, to access a data item child we use the field name. The separator is `/`. In data vectors we can access vector items with indexing, the syntax is `#` sign concatenated with a non-negative integer. To access a data atom from a data item the same method is used: separator plus fieldname. Accessing a parent is possible with the `..` path expression. Examples with these type definitions:

- `/`: access the root vector
- `/TestIC`: access the project called "TestIC"
- `/TestIC/schematic/components/C0/rot`: access the rotation angle of the component called *C0* in the schematic of project *TestIC*. Name *C0* is given by the **xJ** naming system since the `component` type defined prefix `C`.

In case of heterogeneous vectors explicit indexing is to be used. The prefix is always specified by the child item because it is an information specific to the child type, not to the vector type. Sub-structures are always *inserted* into data vectors. The program's data structure can be extended only this way .

In **xJ** we use XSL files to provide data import and export from/to any format. The coder doesn't have to maintain persistence code segments because **xJ** manages the synchronization between data structures in the memory and XML files on disk. **xJ** also provides several extra features to make data structure creation easier: custom atomic types (for example value range for a floating point field), description property for all types, shadow vector mechanism with filtering to access certain parts of the data structure through another view and persistent data fields, which keep their values between program sessions.

In a program built with the **xJ** library functions can be called from three different sources:

- *Constructors*. After overriding the default constructor during object creation the JAVA virtual machine executes the constructor code.

- *Data handlers*. After registering data handlers (mainly in constructors) the corresponding callbacks are called by **xJ** on data access. These handlers have several user areas:

- Check conditions before value setup or insertion.
- Do post operations (example: synchronizations) after data access.
- Manage cache variables which may speed up user interface or calculations.

Data handlers are registered to object and field name pairs, this way extending the appropriate handler class. The class has a variable called `THIS` which is set up to the calling before it starts any handler function. This allows using the same handler instance for several objects.

- *User interface event handlers*. User interactions can also trigger code segments.

Graphical editors are structured and defined in XML in the same way as data structures. Every built-in data atom (like a single number) has a corresponding atomic editor. Synchronization tasks – such as editor refreshing, field value dependencies – are managed by **xJ**.

Package `xj.ui` contains classes related to user interface development. These editors edit data atoms, rather than data items. But an abstract class handles only data items.

This contradiction is resolved by the data structures and data handlers. Every data atom has a data item parent. When a data atom changes it means that one of the parent's `setX()` methods was called. The parent administrates this as its own data change and notifies the registered editors. We set the parent data item as focus item and pass the field name of the data atom to the editor. This is also useful for using the same editor set for different data structures at different times. This is the key of the vector editor.

The persistence support in JAVA is extended by **xJ** to provide static data fields. **xJ** registers all used data types and instances, allows the programmer to access preference variables as easy other values and guarantees that if any static field has been changed all related class instances will be notified and the corresponding data event handler functions will be called.

With these programming techniques software development is reduced to modeling in **xJ**'s XML and JAVA language. This guarantees the easy maintenance of the program code. Besides the reasons described so far, we have chosen JAVA due to the following:

- platform independence is guaranteed, no additional developer resource needed to keep the source portable,
- the clarity of the language forces the programmer to create the program from the model,

- reflection support which is essential for **xJ**'s on the fly type management,
- easy debugging,
- additional *portable* features: persistence support, networking, image processing, model oriented 3D programming,
- professional tools for XML,
- possibility of preparing demos on the web from the same code.

Most of these advantages listed above are not accessible in other programming environments or are not stable enough according to our experience.

Traditional object oriented data structure design would have requested lots of similar code segments to manage the heterogeneous data structures: *get()/set()* methods, *load()/save()* functions and import/export functions for each solver file format. The data event based model of **xJ** eliminates these tasks on the cost of a small overhead in execution.

There is no doubt that JAVA programs are slower than native codes. In Sissy the all solvers (written in C/C++) are strictly separated from the GUI. That is why lower performance of the graphical user interface is not relevant in this simulation system. The connection between the user interface and the solvers is provided by the Sissy Solver Server (SSS).

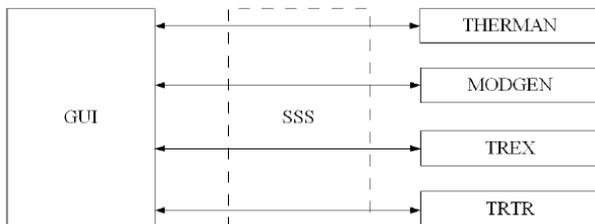


Figure 3: The structure of the solver server

4. THE SSS ARCHITECTURE

In the recent *Sissy* implementation we split the program into two parts, based on the classical thin client model: the *Graphical User Interface* manages problem definition, simulation flow control and the *Sissy Solver Server* connects multiple *GUI* clients and the solver programs. The following figure illustrates the separation of the user interface and solver programs connected through the SSS module (Figure 3). This method has several advantages:

- Running SSS on a fast server computer and using *GUI* client on slower computers results in better resource management. While simulations run in background, the designers can work on other tasks.
- A company may provide SSS service on the web on commercial basis. The vendor has to develop and

maintain only the solver programs and keep them on the server side. The customers do not have to upgrade their simulation systems with newer solver releases.

The SSS protocol has version checking which controls the communication in three levels:

- *Protocol version*. This changes when new commands have been introduced into the SSS protocol. The *GUI* must check this version on new connections.
- *Solver version*. Describes the state of the solver program group. If any solver component changes this number should indicate that change.
- *Little/big endian*. Since both *GUI* and SSS are platform independent they have to check byte order before binary transfers. SSS determines the byte order, the *GUI* performs necessary conversions.

There is also the possibility of *standalone installation*: install server and client transparently on the same computer. The user feels that (s)he runs only one CAD program. This allows vendors to offer the *Sissy* system in standalone or client-server versions.

Applet version of *Sissy* would be impossible without SSS. Creating small JAVA applets with prepared examples (preloaded templates) is good for demonstration.

5. THE SISSY SIMULATION SYSTEM

The simulation system consists of the following parts:

- *CTM (Compact Thermal Model) library editor*. *Sissy* is the first trial implementation of a new XML based neutral semiconductor package compact thermal model (CTM) library file format that has been recently proposed in the JEDEC JC15.1 committee. In CTM library editor mode of *Sissy* the project contains one or more compact thermal model libraries. Each library contains general information (name, description, version, creator, date, etc.) and package types with the following components:
 - general settings (e.g name, description)
 - physical structure: description of the IC package geometry (like outline, footprints, surfaces)
 - the compact thermal model of the package itself in form of a thermal "netlist".
- *Thermal structure editor and simulator*. This project contains all information needed to perform thermal simulations of an IC chip, MCM or printed circuit board:
 - material catalog
 - CTM library (if PCB-s are analyzed)
 - Heat sink library if CTM library is present
 - Physical structure: 3D arrangement (layout) of dissipating elements, boundary conditions
 - Analysis conditions

- *Electrical network editor and circuit simulator.* The skeleton of a circuit simulation project tree is the following:
 - Semiconductor library (semiconductor catalogs) with a link to CTM library items in case of discrete semiconductors to provide full set of information for board-level simulation (electrical parameters of chips plus package thermal model). Instances in this library are sets of parameters for the different semiconductor model equations of the circuit simulation program in use.
 - Component library: set of component types and models. Components in the network schematic are instances of models. Types are only groups of models, like Gummel-Poon model for BJT-s. Model instances may refer to different parameter sets defined in the semiconductor library. Since the component library is editable it is easy to interface any circuit simulator to the system by changing the set of component supported by the simulator used.
 - Circuit schematic: built of instances of different elements of the component library
 - Analysis conditions
 - Electro-thermal system editor and simulator. The electro-thermal project is the union of the previous two projects, except analysis settings, providing means of editing both the physical structure and the circuit schematic (Figure 4).

In an analysis session the GUI creates an XML description of the necessary data structures, transforms it into the input file formats of the solver through XSL files, transfers input files to the *SSS* and instructs *SSS* to run the required solver programs, checks existence of the results and retrieves

them and finally parses and displays the simulation results in the format requested by the user.

6. CONCLUSIONS

In this paper we presented a novel solution in software technologies – an extension of the JAVA language – that allowed a clear, platform independent and easy-to-maintain implementation of an electro-thermal simulation system that supports both chip and board level simulation. For board level simulation we realized the first pilot implementation of a neutral, vendor independent semiconductor package compact thermal model library format based on XML. The Sissy Solver Server allows decoupling the actual simulator solvers used and the graphical user interface and supports using a heterogeneous computer environment,

7. REFERENCES

- [1] V. Székely: Accurate calculation of device heat dynamics: a special feature of the TRANS-TRAN circuit analysis program, *Electronics Letters*, vol. 9, no. 6, pp. 132--134, 1973.
- [2] V. Székely, A. Poppe, M. Rencz, A. Csendes, A. Pahi: Self-consistent electro-thermal simulation: fundamentals and practice, *Microelectronics Journal*, vol. 28, pp. 247--262, 1997.
- [3] V. Székely, A. Poppe, M. Rencz, M. Rosental, T. Teszéri: THERMAN: a thermal simulation tool for IC chips, microstructures and PW boards. *Microelectronics Reliability*, Vol. 40, pp. 517-524, 2000
- [4] M. Kuuse, M. Loikannen, G. Bognar: Theoretical investigation of thermal feedback effect in low-power circuits, In this volume
- [5] A. Poppe: Draft for a standardized description of dynamic CTM-s of semiconductor devices packages for board level thermal simulation

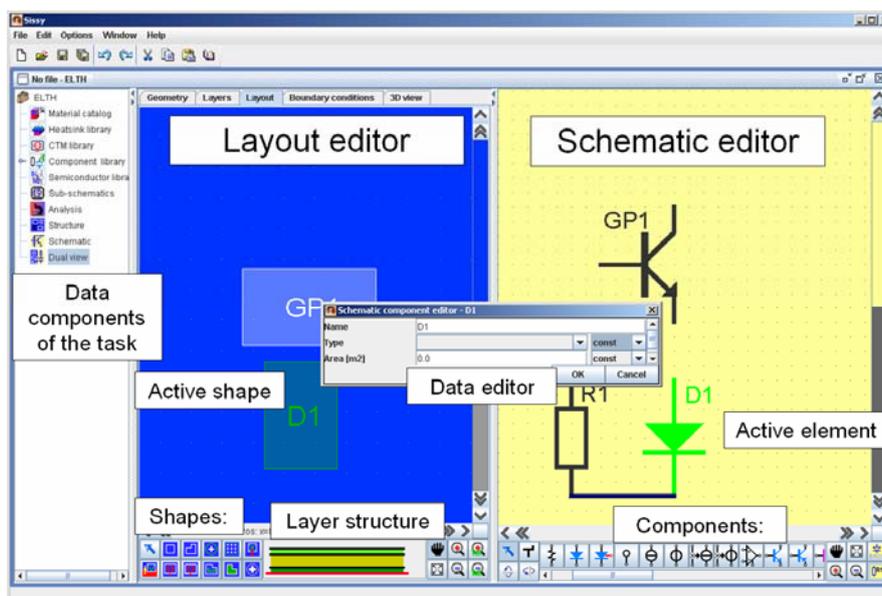


Figure 4: Sissy project bar and editors in case of an electro-thermal project