Hardware-Software Co-Visualization: Developing Systems in the Holodeck

Rolf Drechsler Mathias Soeken

Group for Computer Architecture, University of Bremen, Bremen, Germany
Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{drechsle,msoeken}@informatik.uni-bremen.de

***Abstract*—Modern systems consisting of hardware and software are becoming more and more complex. The underlying data of next generation systems will consist of billions of entries in terms of components or lines of code. Handling this data efficiently is one of the major challenges for future EDA. In order to provide a meaningful preparation for these complex issues it is inevitable to deal with highly elaborated visualization techniques. It is unimaginable how data sets of this size could be grasped without advanced plotting methods.**

Although a lot of effort has been put into research for visualization of hardware and software, techniques hardly exist that consider them in combination. Besides that, in most cases visualization techniques concentrate on the illustration of the system's structure and behavior, e.g. to ease debugging. However, far more information can be integrated. As an example, in the context of verification the accentuation of coverage metrics on top of the structural visualization of a system would immediately pinpoint the verification engineer to areas that are poorly validated. Furthermore, when considering the co-design of hardware-software systems, design exploration can be carried out much easier when the designer gets immediate visual feedback.

Inspired by recent achievements in visualization methods and the invention of sophisticated machinery, in this invited paper we propose the use of *Hardware-Software Co-Visualization* (HSCV). The potential of current techniques as well as their limitations will be demonstrated. Furthermore, we are seeking for alternative methods in system visualization that go beyond monitors and printed pages. Techniques from 3D rendering and virtual reality are utilized for this purpose leading to a holistic environment in which complex systems can be grasped within seconds just as huge data sets in the context of plots. State-of-the-art is presented and directions for future work are outlined.

I. Introduction

Algorithmic visualization has its roots in flowcharts initially been proposed in [1]. Since then a vast amount of approaches to visualize both complex hardware and software systems has been presented for an ease of comprehension. With growing complexity of the developed systems also requirements for the visualization methods increase. As an example, it is not practical to visualize the precise data flow of modern processors using flowchart diagrams.

Independently from each other various approaches have been presented that either visualize hardware or software systems emphasizing on metrics that are particularly important for the respective area. However, since in modern systems the borders between hardware and software are rather fluid, new visualization techniques are required that take both hardware and software aspects into account. Moreover, besides the structure and the behavior of a system, also the integration of quality metrics into the visualization is of importance.

In this invited paper, we present a vision for *Hardware-Software Co-Visualization* based on ideas from 3D software visualization. We have implemented our ideas based on SystemC TLM designs that are used for modern systems containing both hardware and software. Furthermore, we envision the use of virtual worlds in order to facilitate an easy orientation in the 3D visualization.

The paper is structured as follows. The next section reviews related work in both hardware and software visualization. Based on the related work, first ideas and results for HSCV are presented in Section III. Afterwards, Section IV provides ideas on how designers can interact with the visualizations through virtual worlds and an implementation is sketched in Section V. Section VI concludes the paper.

II. Related Work

Visualization is an intensively studied field, however, until now hardware and software systems have mainly been considered separately.

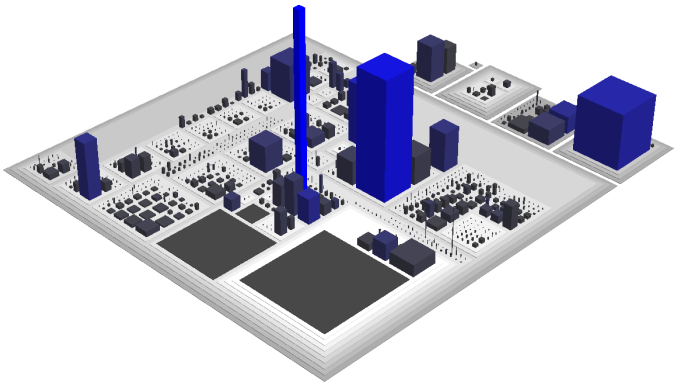

Fig. 1. CodeCity

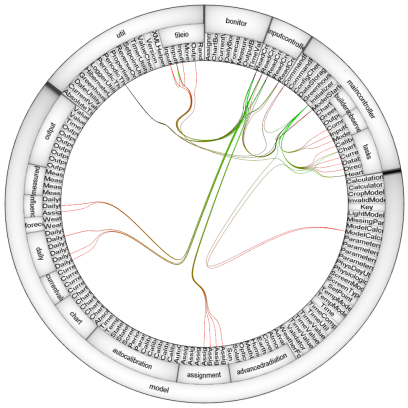

Fig. 2. EXTRAVIS

In [2] an approach called CODECITY has been proposed in which large software libraries are visualized as cities. Each class represents a building in a city and its dimension is determined by its number of methods (height) and its number of attributes (width and length). Classes from the same package are grouped within a district in the cities; subpackages form subdistricts and so on. However, connections between the classes have not been taken into account. Fig. 1 shows the visualization of the Java IDE *jEdit*.

The approach presented in [3] called EXTRAVIS (cf. Fig. 2[1]) focuses on visualizing the dependencies between software components. In a large two-dimensional circle [4] all structural entities are placed at the rim in different dimensions according to their size. Relations between the entities are emphasized by lines which thickness provides information on how tight they are coupled, however, they do not give detailed information about the data flow.

Approaches for hardware visualization differ depending on the abstraction level they are targeting. At lower abstraction levels the efficiency of the algorithms is a primary cost criteria [5]. Two-dimensional visualization techniques for SystemC models have been presented in [6] and [7] and follow the ideas for visualization at the register transfer level. A typical hardware visualization of a design at register transfer level is illustrated in Fig. 3[2].

III. HARDWARE-SOFTWARE CO-VISUALIZATION

In this section, we describe how the different visualization approaches that have been proposed in the past (cf. previous section) can be combined in order to visualize the interaction between hardware and software, e.g. in embedded or cyber-physical systems.

Visualization techniques are significantly important when considering the flood of data we are facing in modern designs. The amount of data is too large that it can efficiently been handled by computers and as a consequence human assistance

[1]Source: http://www.win.tue.nl/~dholten/extravis/text_visible.png
[2]Source: http://www.concept.de/img/rtl-debugger-and-rtl-viewer-I.gif

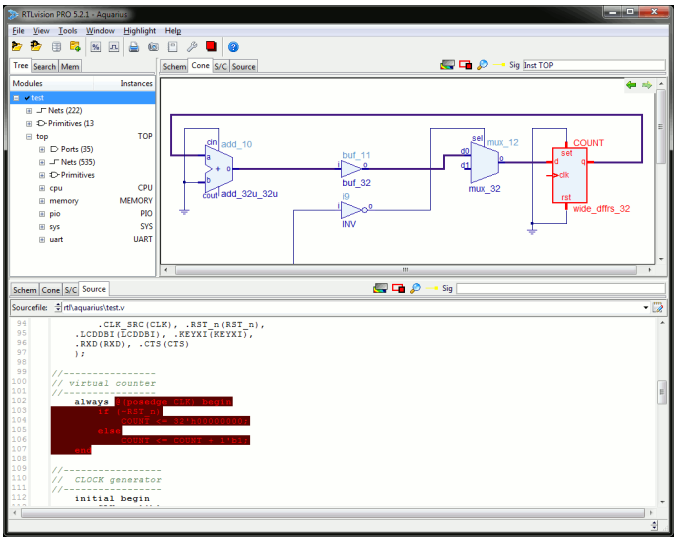

Fig. 3. RTLVISION

is required. *Visual analytics* describes techniques that seek to provide people with better and more effective ways to understand and analyze large data sets [8]. Some information is impractical to measure using a computer, e.g. irregular clusters of data that appear at different spots, which can however be grasped efficiently assuming that the data is visualized appropriately.

Besides the amount of data the multi-dimensionality is presenting a challenge. In the best case all relevant information should be visualized at the same time including

- *structural information* such as lines of code, number of signals, number of attributes, number of methods, and connectivity;
- *behavioral information* such as execution times and simulation traces; and
- *quality metrics* such as complexity, maintainability, test and verification coverage.

But even with three-dimensional visualization schemes incorporating all this information is not easy. As a consequence, we are envisioning to use dynamic visualization techniques besides static ones as reviewed in Section II. As an example, the behavior of a system can be visualized by dynamic techniques. Taken the city metaphor, simulation traces can be visualized by highlighting the building according to their activity and the data flow. Techniques from visual analytics can help to immediately determine irregular behavior that is hard to detect for a computer program.

To visualize both hardware and software components we make use of both sides of the base area. Everything above the ground is considered software and everything below represents hardware. Coverage information can be visualized by coloring the progress onto the buildings.

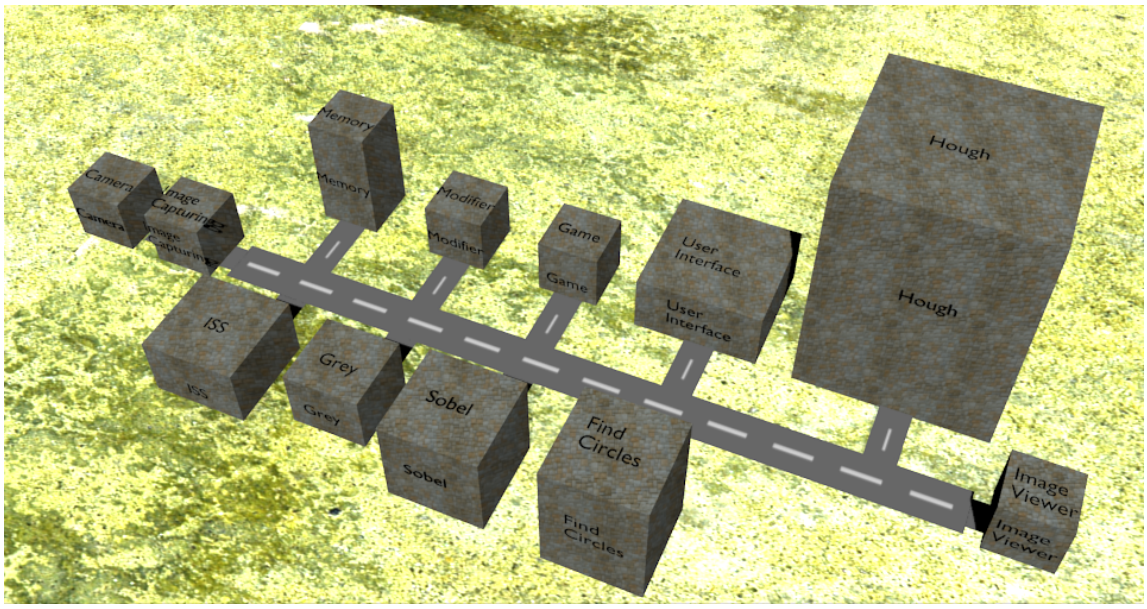

Fig. 4. City visualization scheme in a virtual world

Hierarchies are implemented by having several cities and districts. Houses can be entered in order to obtain more information about a certain module. The different processes and methods can be visualized by means of stories or rooms. In principle, the multi-dimensionality can also be achieved by offering different views that focus on different attributes.

Example: We want to illustrate this by means of an example. The city visualization scheme can be transferred to ESL designs by adjusting the considered metrics. Instead of simple code metrics such as the number of lines or the number of methods we are rather interested in more advanced quality metrics as listed above. In particular, we are considering complexity and maintainability as quality attributes. In software, condition complexity [9] e.g. measures the number of linear independent paths in a program. For hardware an entropy-based concept has been presented in [10]. In our implementation we target SystemC TLM models, therefore, we use a local metric that we apply to each SystemC module. Our metric utilizes the size of the TLM payload data, the memory allocated within a memory, and the operators used in all member functions.

The goal of maintainability is to reflect the adaptability and modifiability of a SystemC module which is required to correct errors or to improve the performance. For this purpose, we are using the *maintainability index* as proposed in [11] and measure it for each single SystemC TLM module.

In general, the measures we are considering in this work are based on attributes which are directly measurable based on the code. There are many other interesting attributes [12] which may be used additionally or as an alternative.

Fig. 4 shows the application of the quality metrics to visualize a SystemC TLM model. Every module represents one building in the city, whereas the height and base area of each building corresponds to the module's complexity and its maintainability, respectively. Furthermore, the connectivity information of the TLM modules are visualized in terms of streets that connect the buildings.

IV. Access Through Virtual Worlds

When implementing the visualization schemes that have been proposed in the previous section by means of a virtual world, further possibilities for the designer to interact with the system can be obtained. By using recent achievements in virtual reality and corresponding hardware devices, it is possible to basically be *inside* the chip. This allows for much better visualization methods and hence a better comprehension. Irregularities in the design can easily be noticed by observing the environment making use of plain intuition, since in a virtual world a complex scene can be grasped within a few seconds, emphasizing the effects of visual analytics.

We make use of state-of-the-art rendering techniques as they can be found in the design of modern 3D computer games. It is already impressive how much more information can be brought to a computer monitor by just displaying the data inside a virtual world in which a character is navigated using simple keyboard and mouse control. In contrast, in some 3D visualization tools it sometimes feels cumbersome to bring the displayed model into the right perspective. We use the city metaphor described in the previous section for an appropriate visualization of the design in the virtual world.

By making use of sophisticated hardware devices the virtual world should actually be entered. There are two important devices to realize this undertaking, namely a hollow sphere to control rotation and direction according to the user's footsteps and a head mounted display to project the virtual world directly to the eyes of the designer.

All together, this allows for a true-to-reality experience enabling to make use of the designer's intuition for some design flow tasks. Also dynamic visualization techniques can easily be implemented in virtual worlds. An ongoing simulation can be visualized while the user is just watching the impact on the scene and detects irregularities by making use of plain intuition when some things "just look wrong."

V. Implementation

Although the idea might seem futuristic, all parts that are needed for an implementation are already existing, and they only need to be put together appropriately. For this purpose, we distinguish between the software side and the hardware side. The software side is responsible for rendering the circuit or system and displaying it in a three dimensional virtual world, whereas the hardware side consists of a collection of devices in order to navigate and control the virtual world.

Development kits such as the *Unreal Development Kit*[3] or the *Unity Game Engine*[4] can be used to create virtual worlds. While this software usually manages the organization of the actual world, the navigation of the character, and the interaction with virtual objects, the creation of 3D assets should be carried out using off-the-shelf 3D modeling software such as Blender[5].

In order to get a true-to-reality experience we are aiming for a solution in which the designer can literally enter the virtual world. For this purpose, we are making use of modern hardware that allows for navigating the virtual character and adjusting the visual viewpoint according to the body position.

An example implementation could consists of a hollow sphere such as the *VirtuSphere* [13] which can be entered by a human and can be rotated in any direction by the user's steps. The information received from this device can be used in order to apply an appropriate transformation to the virtual world thereby enabling the feeling that one can move around in the world like in a real world.

To amplify the feeling of being in the virtual world that is displaying the circuit or system being designed, one can make use of head mounted displays [14] instead of conventional computer monitors. As a consequence, the virtual world is directly displayed in front of the eyes and since one monitor is used for each eye also a more realistic 3D visualization can be achieved.

Using such an information enhances the access to all relevant information. As a result, the effects described by visual analytics are emphasized, therefore enabling to solve design flow tasks that currently cannot be handled automatically by a computer alone.

[3] www.unrealengine.com
[4] www.unity3d.com
[5] www.blender.org

VI. Conclusion

In this paper we have presented Hardware-Software Co-Visualization. For this purpose, we have reviewed state-of-the-art visualization techniques and described how they can be used and extended to visualize systems that contain both hardware and software. Visualization should help to manage challenges that cannot be handled by the computer alone anymore. Techniques of visual analytics are taken into account in order to make use of human abilities to detect e.g. irregular patterns in the behavior. We have discussed further metrics that should be taken into consideration that go beyond structural and behavioral information. Furthermore, we illustrated how the implementation in a virtual world can enhance the access to all relevant information.

Acknowledgments

The authors thank Daniel Große and Marc Michael for helpful discussions and their support in preparing this manuscript and implementing the prototype.

References

[1] H. H. Goldstine and J. von Neumann, "Planning and coding of problems for an electronic computing instrument," Tech. Rep. Part II, Volume II, 1947.

[2] R. Wettel and M. Lanza, "Program comprehension through software habitability," in *Int'l Conf. on Program Comprehension*, Jun. 2007, pp. 231–240.

[3] B. Cornelissen, A. Zaidman, D. Holten, L. Moonen, A. van Deursen, and J. J. van Wijk, "Execution trace analysis through massive sequence and circular bundle views," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2252–2268, Dec. 2008.

[4] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 741–748, Sep. 2006.

[5] R. Drechsler, W. Günther, T. Eschbach, L. Linhard, and G. Angst, "Recursive bi-partitioning of netlists for large number of partitions," *Journal of Systems Architecture*, vol. 49, no. 12–15, pp. 521–528, Dec. 2003.

[6] D. Große, R. Drechsler, L. Linhard, and G. Angst, "Efficient automatic visualization of SystemC designs," in *Forum on Specification and Design Languages*, Sep. 2003, pp. 646–658.

[7] C. Genz, R. Drechsler, G. Angst, and L. Linhard, "Visualization of SystemC designs," in *Int'l Symp. on Circuits and Systems*, May 2007, pp. 413–416.

[8] D. A. Keim, L. Zhang, M. Krstajic, and S. Simon, "Solving problems with visual analytics: Challenges and applications," *JMPT*, vol. 3, no. 1, pp. 1–11, 2012.

[9] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, Dec. 1976.

[10] B. Menhorn and F. Slomka, "Design entropy concept: a measurement for complexity," in *Int'l Conf. on Hardware/Software Codesign and System Synthesis*, Oct. 2011, pp. 285–294.

[11] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and application of an automated source code maintainability index," *Journal of Software Maintenance*, vol. 9, no. 3, pp. 127–159, May 1997.

[12] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston, MA, USA: PWS Publishing Co., 1998.

[13] E. Medina, R. Fruland, and S. Weghorst, "VIRTUSPHERE: Walking in a human size VR "hamster ball"," *Human Factors and Ergonomics Society Annual Meeting*, vol. 52, no. 27, pp. 2102–2106, Sep. 2008.

[14] T. Shibata, "Head mounted display," *Displays*, vol. 23, no. 1–2, pp. 57–64, Apr. 2002.