COMPUTING SCIENCE

Resource-driven Modelling of Complex Digital Systems with Uncertainty

Ashur Rafiev, Alexei Iliasov, Alexander Romanovsky,
Andrey Mokhov, Fei Xia and Alex Yakovlev

TECHNICAL REPORT SERIES

No. CS-TR-1413 February 2014

TECHNICAL REPORT SERIES

No. CS-TR-1413 February, 2014

Resource-driven Modelling of Complex Digital Systems with Uncertainty

A. Rafiev, A. Iliasov, A. Romanovsky, A. Mokhov, F. Xia and A. Yakovlev

Abstract

ArchOn is a new architecture-open, resource-driven, and layer-agnostic modelling method for large complex computing systems focusing on system performance and survivability metrics, including speed, throughput, and crucially, reliability. This paper describes first exploratory uses of ArchOn to model system operations under uncertainties in both the environment and system components. The resource-driven nature of ArchOn allows the modelling of uncertainty in terms of disruptions in the resource dependencies, which can relate to such real-world issues as single event upsets (SEUs) in hardware. A system solving a typical image processing problem serves as the example target system for this exploration.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

RAFIEV, A., ILIASOV, A., ROMANOVSKY, A., MOKHOV, A., XIA, F., YAKOVLEV, A.

Resource-driven Modelling of Complex Digital Systems with Uncertainty
[By] A. Rafiev, A. Iliasov, A. Romanovsky, A. Mokhov, F. Xia, A. Yakovlev

Newcastle upon Tyne: Newcastle University: Computing Science, 2014.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1413)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1413

Abstract

ArchOn is a new architecture-open, resource-driven, and layer-agnostic modelling method for large complex computing systems focusing on system performance and survivability metrics, including speed, throughput, and crucially, reliability. This paper describes first exploratory uses of ArchOn to model system operations under uncertainties in both the environment and system components. The resource-driven nature of ArchOn allows the modelling of uncertainty in terms of disruptions in the resource dependencies, which can relate to such real-world issues as single event upsets (SEUs) in hardware. A system solving a typical image processing problem serves as the example target system for this exploration.

About the authors

Ashur Rafiev is an RA on the EPSRC PRiME Program Grant. In this project he leads the development of the ArchOn modelling environment.

Alexei Iliasov is a Researcher Associate at the School of Computing Science of Newcastle University, Newcastle-upon-Tyne,UK. He got his PhD in Computer Science in 2008 in the area of modelling artefacts reuse in formal developments. His research interests include agent systems, formal methods for software engineering and tools and environments supporting modelling and proof .

Alexander (Sascha) Romanovsky is a Professor in the Centre for Software and Reliability, Newcastle University. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring and verification of fault tolerance. He received a PhD degree in Computer Science from St. Petersburg State Technical University and has worked as a visiting researcher at ABB Ltd Computer Architecture Lab Research Center, Switzerland and at Istituto di Elaborazione della Informazione, CNR, Pisa, Italy. In 1993 he became a postdoctoral fellow in Newcastle University, and worked on the ESPRIT projects on Predictable Dependable Computing Systems (PDCS), Design for Validation (DeVa) and on UK-funded projects on the Diversity, both in Safety Critical Software using Off-the-Shelf components. He was a member of the executive board of EU Dependable Systems of Systems (DSoS) Project, and between 2004 and 2012 headed projects on the development of a Rigorous Open Development Environment for Complex Systems (RODIN), and latterly was coordinator of the major FP7 Integrated Project on Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). He now leads work on fault tolerance in Systems of Systems within the COMPASS project and is Principal Investigator of Newcastle's Platform Grant on Trustworthy Ambient Systems.

Andrey Mokhov studied computing science at Kyrgyz-Russian Slavic University from 2000 to 2005. After graduation with honours he joined the Asynchronous Research Group at Newcastle University as a PhD student and in 2009 he successfully defended his PhD dissertation. Currently he is a research associate in the School of Computing Science, Newcastle University. His research interests include different levels of electronic design automation: from formal models for system specification and verification to logic synthesis and application-specific optimisation.

Fei Xia is a Senior RA on the EPSRC PRiME Program Grant. Fei is with the EE School. His research interests include Asynchronous Data Communication. Asynchrnous System Design. Systems and Networks on Chip. Energy and Power in Computing.

Alex Yakovlev received D.Sc. from Newcastle University in 2006, and M.Sc. and Ph.D. from St. Petersburg Electrical Engineering Institute in 1979 and 1982. Since 1991 he has been at the Newcastle University, where he worked as a lecturer, reader and professor at the Computing Science department until 2002, and is now heading the Microelectronic Systems Design research group (http://async.org.uk) at the School of Electrical, Electronic and Computer Engineering. His current interests and publications are in the field of modeling and design of asynchronous, concurrent, real-time and dependable systems on a chip. He has published four monographs and more than 200 papers in academic journals and conferences, has managed over 25 research contracts.

Suggested keywords

MANY-CORE
RELIABILILY
MODELLING
ANALYSIS
FAULT ASSUMPTIONS

Resource-driven Modelling of Complex Digital Systems with Uncertainty

Ashur Rafiev, A. Iliasov, A. Romanovsky
School of Computing Science
Newcastle University, UK
{ashur.rafiev, alexei.iliasov, alexander.romanovsky}@ncl.ac.uk

A. Mokhov, F. Xia, A. Yakovlev
School of Electrical and Electronic Engineering
Newcastle University, UK
{andrey.mokhov, fei.xia, alex.yakovlev}@ncl.ac.uk

ABSTRACT

ArchOn is a new architecture-open, resource-driven, and layer-agnostic modelling method for large complex computing systems focusing on system performance and survivability metrics, including speed, throughput, and crucially, reliability. This paper describes first exploratory uses of ArchOn to model system operations under uncertainties in both the environment and system components. The resource-driven nature of ArchOn allows the modelling of uncertainty in terms of disruptions in the resource dependencies, which can relate to such real-world issues as single event upsets (SEUs) in hardware. A system solving a typical image processing problem serves as the example target system for this exploration.

1. INTRODUCTION

Translating integration scaling to performance growth is challenged by such factors as the utilization wall [6]. Processor clock frequencies have not increased since 2005 despite increasing transistor speed [5]. Using multi-core to maintain the predictions of Moore's Law [4] will only delay the inevitable [3], as the near-threshold computing (NTC) advantages are limited by such factors as variability and reliability concerns when voltage is radically scaled down. When other issues, such as reliability, are considered in addition to performance in the context of increasing parallelism (e.g. increasing the number of cores), there is little concrete research result to be found in the literature and the picture is even more uncertain.

Modern computing systems, especially mobile and/or embedded systems, must deal with a high degree of uncertainty from within the systems themselves and without in the environment. Examples include the not-always predictable inherent physical parameters such as temperature, voltage, energy availability, external noise, variability etc. and from unpredictable user demands.

This task requires appropriate means to reason about this vaguely defined exploration space. The modelling and simulation methods are required to be open-ended with the capability of easily extending the simulated platform's functionality. We can't rely on some fixed architecture if we want to develop a universal solution. In addition, since the interest of our research includes reliability analysis, our models require formal specification and verification, which can't be done by simulations alone. Another important requirement for the framework is the ability to operate at different layers of abstraction. Unlike conventional ideas of separating concerns cutting between the layers, our goal is to observe the system as an intricate conglomeration of elements of different nature.

The ultimate goal of modelling is to provide convenient ways of studying complex systems. Graph-based models have been used in the human endeavour to reason about the world around us [2], because most people respond well to graphically represented concepts. This popularity in turn caused a rich set of mathematical techniques to develop within and around graph theory, which has become a very powerful methodology for rigorous reasoning [7, 8]. For example, Petri Nets [9] and similar formalisms support modelling for design and analysis with extensive tool support. However, complex systems with many levels of abstraction tend to present difficulties to these existing methods, especially when cross-layer behaviours need to be investigated. Some researchers use intermediate frameworks, so that the process of modelling becomes clear and insightful whilst still preserving a formal background [10].

ArchOn was developed to meet these challenges [11]. This paper describes the fundamentals of ArchOn, including its resource-driven approach, which makes it straightforward to reason about entities that can be regarded as resources (hardware, software, energy, time, etc.), and its graphical representation for convenience. Previously, ArchOn has only been used to reason about energy, performance and concurrency. In this work we start to explore uncertainty, including how it can be represented within the ArchOn framework and what properties can be derived from simulating these models.

This paper is organised as follows: Section 2 outlines the ArchOn modelling framework. Section 3 proposes methods with which uncertainty, including SEUs, may be represented in ArchOn and how such representations relate to real-world issues. Section 4 gives a use-case example and outlines the plans for future work. As this is on-going research, the results presented here are subject to future revisions and developments.

2. ARCHON ENVISIONED

The central subject of our method is the study of a computational platform comprising a number of diverse resources and the way resources may be handled in order to realise a computation. A resource is in this case an indivisible element required by the system in order to change its state, and it is defined by its function and availability in relation to this transition.

We propose to represent a system with a relation graph, consisting of a set of vertices and a set of edges. Each vertex represents a single resource and each edge represents a dependency between two resources. We also view the system as a dynamic set of resource relations: resources may become unavailable in certain points in time, and the model must be able to capture this behaviour. A dynamic model can be represented using the state transition semantic, where the states are concrete resource allocations or configurations.

One of the applications of this approach is simulating hardware running a software. In this case, the resources are concrete hardware units like ALU, MMU, etc. Resource states store unit data, and the resource dependencies represent data transfer between the units. During the simulation we can introduce faults: invalid set or unset arcs in the resource graph. These might correspond to various hardware mishaps, we prefer to see them at the more abstract level as resource relations not being ensured.

While building the simulation tool, in order to understand the interaction between the modules, we have taken an unusual approach: "think hardware – make software". We imagined that

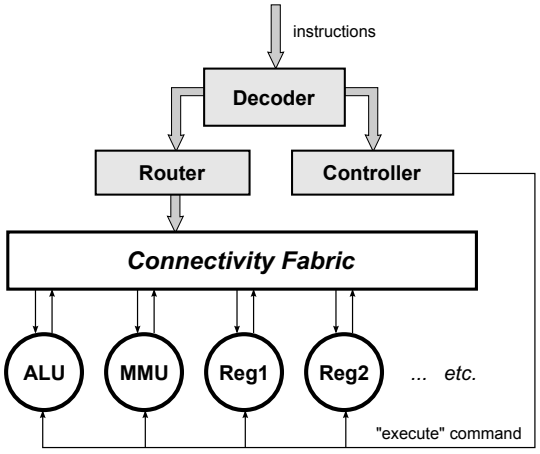

Figure 1: What a flexible architecture would look like in silicon? Making software design decisions while thinking in hardware terms.

we are to make a flexible hardware architecture. What would the challenges be, and how should we overcome them? Just as FPGA allows customisation at the scale of logic gates, our hypothetical hardware must allow customisation at the scale of hardware modules. Although in our research we do not have an actual task to deliver such a platform, this non-traditional thinking immediately paid back with a number of original design decisions, giving a better insight to the simulation software.

Figure 1 shows a communication-based hardware architecture that could potentially emulate most cyber-physical systems. This type of architecture is called transport-triggered architecture [1]. It hasn't become popular in general purpose microprocessors, but it appeared attractive for our purposes. Assuming that the target system has an instruction set, its software can be recompiled into the connectivity fabric routing commands. The process of executing such software would have alternating phases of configuring the connectivity fabric and executing modules.

3. MODEL FUNDAMENTALS

Putting together the outlines described in the previous section, we formally define *platform architecture* as a labelled directed graph $\mathscr{A} = (\mathscr{V}, \mathscr{E}, \mathscr{X}, \Phi)$, where $\mathscr{V}$ is the set of all platform resources, and $\mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}$ captures all possible (allowed) dependencies between them; $\mathscr{X}$ is a set of labels that can be assigned to vertices and edges in various ways by label assignment functions $\phi : \mathscr{V} \cup \mathscr{E} \to \mathscr{X}$, $\phi \in \Phi$. For each resource $v \in \mathscr{V}$ we also define: W_v – the set of possible resource states (can be infinite), and f_v – its node function, explained later.

An architecture is a loose, overarching agglomeration of concrete configurations. During the lifetime of an architecture instance we can observe the switching of resources, dependencies, and labelling. A configuration is understood to be a sub-graph $G = (V, E, X, \phi)$ of $\mathscr{A}$, where $V \subseteq \mathscr{V}$ is a set of allocated resources, $E \subseteq \mathscr{E}$ are active dependencies between them. Only one way of labelling $\phi \in \Phi$, $X \subseteq \mathscr{X}$ is allowed per configuration.

This is not sufficient to describe the whole state of the system though, as the resources may change their internal state over time as well. The state of the system is defined by a tuple (G, U), where G is a resource dependency graph, and U is a resource state vector giving for each $v \in G$ its state $u_v \in W_v$. We know, however, that the resource graph state space $\mathscr{G} = (G_0, G_1, \ldots, G_N)$ is finite for a finite architecture $\mathscr{A}$, while the set of different resource states $\mathscr{U} = (U_0, U_1, \ldots)$ may be unbounded. The state space of the system is a Cartesian product $\mathscr{G} \times \mathscr{U}$ giving an infinite state-transition machine, which is not convenient for a model. Therefore, we want to study two parts separately:

1) *Resource graph evolution* is a top level transition system that works on resource dependency graphs. This can be considered as an FSM where graphs represent states. Transitions between these graphs do not change the state of resources: $(G, U, \triangle) \to (G', U, \triangledown)$.

2) *Resource state evolution* is an inner transition system that operates on resource states: $(G, U, \triangledown) \to (G, U', \triangle)$.

Changing between $\triangle$ and $\triangledown$ means that both evolutions are alternating (as envisioned by hardware routing and execution phases in the previous section).

Transitions between resource states are defined using *node functions*. For each $v \in G$, the node function is defined as $f_v : (G, U) \to U$. Typically, the node function operates on the sub-graph of G (e.g. the node's pre-set and post-set) and related projection of the vector U.

Algorithm 1 Pseudo code for 3×3 convolution filter

```
0   init ptrSrc, ptrOut, npixels
1   for (; npixels > 0; npixels = npixels − 1) {
2     ptrMask = 0
3     offs = 0
4     res = 0
5     for (dy = 3; dy > 0; dy = dy − 1) {
6       for (dx = 3; dx > 0; dx = dx − 1) {
7         a = M[ptrMask]
8         ptrMask = ptrMask + 1
9         b = M[ptrSrc + offs]
10        res = res + a · b
11        offs = offs + 1
12      }
13      offs = offs + 256 − 3
14    }
15    M[ptrOut] = res
16    ptrOut = ptrOut + 1
17    ptrSrc = ptrSrc + 1
18  }
```

4. USE CASE EXAMPLE

The method to supply resource graphs to the simulation software has been derived from our hardware vision, described in Section 2. As in Figure 1, we view the simulator modules (resource nodes) as connected via the connectivity fabric, and the simulator input parser works as a router. Table 1 shows some commands for this "router", which provide step-by-step graph configurations as well as explicit invocations of resource state transitions. We call it *graph assembly language*. Fault injection is done at the level of graph assembly commands: at the given error rate some commands are skipped. Possible interpretations are also given in Table 1.

As an example we simulate an ARM-based system running a image processing using a convolution algorithm. Given two 3×3 matrices, the goal is to multiply them element-wise and sum up the results. Usually, one of the matrices is a 3×3 sub-matrix of an image being processed and the other matrix, called a *kernel* or a *mask*, represents the required image transformation, e.g., sharpening or edge detection. This step is applied to all 3×3 sub-matrices of a given image, each time producing a value for a pixel in the resulting image. The implementation working on a 256×256 grey-scale image is shown in Algorithm 1. Memory access using pointers is denoted as $\mathbf{M}[ptr]$.

This pseudo-code has been translated into ARM assembly language. For every ARM instruction we derive a resource evolution and translate it into graph assembly language. This is a routine task since all instructions follow a common pattern. The process of translation can be done automatically. For instance, line 10 of the algorithm compiles into `MLA r8,r9,r10,r8`, where registers `r8,r9,r10` correspond to the variables res, a, b respectively. This ARM instruction in graph assembly language is shown

Table 1: Some commands of graph assembly language and interpretation of associated faults

command	description	if missed: possible HW failure
a $\rightarrow$ b	set a dependency between resources *a* and *b*	data-path failure due to early clock
a $\overset{x}{\rightarrow}$ b	set a labelled dependency between resources	data-path failure due to early clock
a $\nrightarrow$ b	unset a dependency	data is late due to a slack in a single path
$G = \emptyset$	clear all dependencies	old data is kept due to a global clock error
go!	"execute" graph: fire all resource state transitions	clock or power gating malfunction
go to X	continue assembly from label X (jump)	program counter error
if condition **go to** X	conditional jump	program counter error or a bit-flipped condition signal

Algorithm 2 ARM instruction `MLA r8,r9,r10,r8` in graph assembly language.

$G = \emptyset$
r9 $\overset{n}{\rightarrow}$ mul
r10 $\overset{m}{\rightarrow}$ mul
go!
$G = \emptyset$
mul $\overset{n}{\rightarrow}$ alu_add
r8 $\overset{m}{\rightarrow}$ alu_add
go!
$G = \emptyset$
alu_add $\rightarrow$ r8
go!

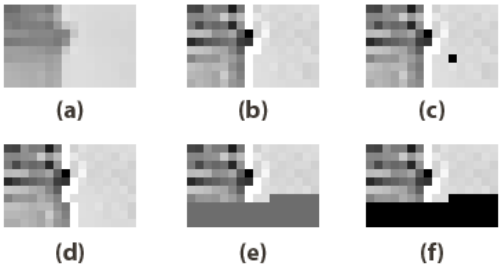

Figure 2: Zoomed fragment of the image and the types of damage: (a) original, (b) "sharpen" filter with no errors, (c) one pixel error, (d) pointer update error (shift), (e) corrupted pointer, (f) premature program termination.

in Algorithm 2.

This sub-instruction level simulation showed the effects of the injected faults with a high precision. As expected, some faults made drastic damage to the output while others were almost unnoticeable. Our method allowed us to pinpoint critical errors and trace them back to ARM instructions and finally the original pseudocode. Figure 2 shows the types of errors observed in the output image. Sub-figures (a) and (b) display the original image and the flawless result. Case (c) error happens when the computation of a single pixel goes wrong, i.e. the fault happens between lines 2 and 15 of the Algorithm 1 – this is the most frequent of errors. In case (d) the pointer increment did not compute due to a fault in the lines 16 or 17. Case (e) also relates to a fault in these lines, but in a more subtle way: a missed "go!" instruction during addition in line 17 causes the adder's previous value, which at the moment equals to *ptrOut*, to be copied into *ptrSrc*. Such an effect can be observed only in a sub-instruction level of simulation, provided by ArchOn. The last case of error, shown in Figure 2(f), is caused by a failure in the main loop condition in line 1 leading to the premature exit from the loop.

In general, we believe the method can be automated and used to analyse large software and hardware systems, much larger than the given toy example. The next step is to provide a method of applying the collected knowledge in order to improve the reliability and survivability of the system. The most straightforward solution may imply duplicating most critical resource relations.

5. CONCLUSION

The ArchOn method is developed to help designers of complex systems with multiple design objectives. It is based on a unique resource dependency graph approach.

As an application of this modelling method, we proposed an experiment of injecting faults at the level of resource dependencies in order to explore the impact on the system's reliability. The given image processing example proved to be quite stable: most frequent errors were localised to a single pixel damage. However, critical points were detected including a very subtle one.

The method can be automated and the collected knowledge can be used to improve survivability of hardware-software systems.

Acknowledgement: This work is supported by EPSRC grants EP/K034448/1 and EP/I038357/1.

6. REFERENCES

[1] H. Corporaal. Design of transport triggered architectures. In *Proc. to Design Automation of High Performance VLSI Systems*, pages 130–135, 1994.

[2] R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[3] H. Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[4] S. H. Fuller and L. I. Millett. Computing performance: Game over or next level? *Computer*, 44(1):31–38, 2011.

[5] N. Goulding-Hotta et al. The greendroid mobile application processor: An architecture for silicon's dark future. *IEEE Micro*, 31(2):86–95, 2011.

[6] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[7] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, University of Newcastle upon Tyne, School of Computing Science, 2003.

[8] A. Mokhov and A. Yakovlev. Conditional partial order graphs: Model, synthesis, and application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.

[9] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, 1989.

[10] I. Poliakov. *Interpreted Graph Models*. PhD thesis, University of Newcastle upon Tyne, School of Electrical, Electronic and Computer Engineering, 2011.

[11] A. Rafiev, A. Iliasov, A. Romanovsky, A. Mokhov, F. Xia, and A. Yakovlev. ArchOn: Architecture-open resource-driven cross-layer modelling framework. Technical Report NCL-EEE-MICRO-TR-2014-184, School of EEE, Newcastle University, January 2014.