Using Activity Metrics for DEVS Simulation Profiling
Alexandre Muzy, Laurent Capocchi, J.F. Santucci

To cite this version:

HAL Id: hal-01315176
https://hal.archives-ouvertes.fr/hal-01315176
Submitted on 12 May 2016

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.
Activity metrics can be used to profile DEVS models before and during the simulation. It is critical to get good activity metrics of models before and during their simulation. Having a means to compute a-priori activity of components (analytic activity) may be worth when simulating a model (or parts of it) for the first time. After, during the simulation, analytic activity can be corrected using dynamic one. In this paper, we introduce McCabe cyclomatic complexity metric (MCA) to compute analytic activity. Both static and simulation activity metrics have been implemented through a plug-in of the DEVSimPy (DEVS Simulator in Python language) environment and applied to DEVS models.

1 Introduction

In [1], the concept of activity is introduced for DEVS models as the number of transition functions executions. This quantitative-activity (QA) metric consists in counting the number of state-to-state transitions in a model over some time interval. Activity metrics can be used to profile DEVS models. Profiling consists of collecting statistics of a program execution. Usually, memory usage, durations, frequency of calls, CPU usage and occupation are collected dynamically during program execution. In DEVS modeling and simulation, profiling can be based on activity tracking metrics. In order to analyze models, DEVS designers can correlate the number of transition functions with the CPU time consumed by a transition function. These metrics are available only during the simulation (simulation activity). However, having a means to compute a-priori activity of components (analytic activity) may be worth when simulating a model (or parts of it) for the first time. After, during the simulation, analytic activity can be corrected using dynamic one.

In this paper, we introduce McCabe cyclomatic complexity metric (MCA) (already discussed in [2]) to compute analytic activity. Both static and simulation activity metrics have been implemented through a plug-in of the DEVSimPy (DEVS Simulator in Python language) environment and applied to DEVS models. DEVSimPy [3] is being developed at the SPE laboratory of the University of Corsica.
and is an open source project under GPL v3 license. The DEVSimPy environment is based on the PyDEVS API [4, 5] and is aims at facilitating researches in the SPE team in order to introduce and to validate new concepts around DEVS formalism.

The paper is organized as follows: In the next section we present the activity metric definitions. Section 3 deals with the implementation of these metrics in a new DEVSimPy plug-in called activity-tracking. The obtained results which highlight the relationship between the different metrics are presented in detail. Finally, some conclusions and perspectives are given.

2 Activity metrics definitions

It is critical to get good activity metrics of models before and during their simulation. These activity metrics can be defined from metrics which have been defined in software engineering. A software metric is usually used to determine the degree of maintainability of software products. However, software metrics may be also used to predict the execution time consumption of functions or methods of an object. In DEVS modeling and simulation context, these kinds of metrics can be employed to evaluate the static and simulation activity of models.

2.1 Analytic activity metrics

Among the list of recommended metrics proposed in most of software engineering (Halstead complexity [6], McCabe complexity [7], Coupling [8, 9], etc.) McCabe Complexity has been chosen because: (i) it is one of the most popular metric in software engineering, (ii) it has strong implications for software testing (iii) it can be used as an estimation of the time required for the execution of the transition functions.

McCabe cyclomatic Complexity (MCC) [7] depends only on the decision structure of a program. The cyclomatic number of a directed graph $G$, where each node corresponds to a block program, is a graph-theoretic complexity:

$$MCC(G) = e - n + p$$

where $e$ is the number of edges of the graph, $n$ the number of nodes of the directed graph, and $p$ the number of connected components (exit nodes). This number depends on the number of linearly independent paths, i.e., to the decision (if statement, conditional loops, etc.) structure of a program.

$MCC$ usually correlates with the amount of work required to test a program, therefore it is used to have a measure of the test complexity of a program. However, in DEVS modeling and simulation context, $MCC$ can be used at atomic function level considering the whole functions as a “block program”. Indeed, the higher the number of independent decision paths, the more the system is expected to be an event hub of high CPU activity: if the $MCC$ of an atomic function is high, there is a significant probability that the time spent in the function execution will be high too. Moreover, the $MCC$ can be considered as a metric which provides useful feedback to the DEVS designers during the modeling phase.

For DEVS model, we can defined the McCabe Activity ($MCA$) of an atomic model $AM_i$ as:

$$MCA_{AM_i} = MCC_{δ_{ext}} + MCC_{δ_{int}}$$

(1)
2.2 Dynamic-based activity metrics

Dynamic-based metrics are considered during the execution of a program (simulation process). Concerning the activity of DEVS models, the CPU user time computed and updated during the simulation can be weighted by the quantitative activity.

In [10], the authors define the Quantitative-Activity (QA) of a system as “the number of discrete-events received by the system, over a simulation time period.” According to [11], a measure of activity can be considered as a measure of information processing by counting over some time interval the number of state-to-state transitions in a model. The QA is a notion defined at the modeling level but quantified (tracked) during the simulation. In [12], the authors give the following definition of the total activity for an atomic DEVS model $AM_i$ in a simulation time interval:

$$QA_{AM_i} = QA^\delta_{int} + QA^\delta_{ext}$$  (2)

where $QA^\delta_{ext}$ (resp. $QA^\delta_{int}$) is the external (resp. internal) activity. The external (resp. internal) activity is defined as a natural number equal to the sum of DEVS external (resp. internal) transitions $\delta_{ext}$ (resp. $\delta_{int}$) execution. In [10], the activity of a coupled DEVS model $CM$ is defined as the sum of the total activity of its $N$ atomic models $QA_{AM_i}$ with $i \in \{1..N\}$:

$$QA_{CM} = \sum_{i=1..N} QA_{AM_i}$$  (3)

Let consider the definition of activity given in equation 2 (resp. equation 3) as the definition of the QA for an atomic (resp. coupled) model.

CPU user time is the time spent on the processor running your program’s code (or code in libraries) while system CPU time is the time spent running code in the operating system kernel on behalf of your program. We are interested here in the CPU user time for an atomic DEVS model $AM_i$:

$$CPU_{AM_i} = CPU^\delta_{int} + CPU^\delta_{ext}$$  (4)

3 Activity metrics implementation

DEVSimPy [3] is an open source project initiated by the modeling and simulation (M&S) team at SPE (Sciences Pour l’Environnement) laboratory of the University of Corsica. The aim of this software is to provide developers a collaborative M&S framework in the Python language. It uses the wxPython library which is a blending of the wxWidgets C++ class library with Python. The DEVSimPy M&S kernel is based on the PythonDEVS [4] API which offers a consistent and coherent set of classes in order to construct a modular system and to achieve its hierarchical simulation. A plug-in manager is proposed in order to expand the functionality of DEVSimPy allowing their enabling/disabling through a dialog window.

Built upon definitions of activity metrics given in the previous section, DEVSimPy implements a new plug-in called Activity Tracking (AT). This plug-in increases the handling of the recent definition of the activity metrics thus opening new perspectives for the use of activity tracking in DEVS formalism. DEVSimPy plug-in AT is generic and can be applied to any DEVS models. It does not require any modification of the DEVS simulation algorithm and does not require any additional methods in DEVS models to operate.

The DEVSimPy AT plug-in works the following way:
• The user enables the plug-in (see Figure 1) and chooses the set of DEVSimPy atomic models for activity-tracking (see Figure 2). Before the simulation, the DEVS models are scanned in a recursive way to collect all atomic models selected by the user in the plug-in interface. The external and internal transition functions of all selected models are decorated with a new method aimed at introducing at AT computation of these functions. A decorator function adds a new attribute to the DEVS object in a dynamic way (offered by the Python language combined with the use of oriented aspect programming) for each transition function. Moreover, knowing the code of the transition functions for each selected atomic DEVS model, the associated MCA metric can be performed before the simulation. In the same way, the coupling metrics can be computed from coupling relationships between models inside all coupled models.

• The user can now performed the simulation of the model during which the QA metric is measured by counting the number of DEVS transition functions executions. While the simulation is running, the plug-in offers dynamically a table resuming the QA, MCA metrics for each tracked model.

4 Experiments and results

The model used for the experiments contains 49 atomic models, 15 coupled models, 203 coupling, and 3 levels of encapsulation between coupled models. It models an asynchronous electrical machine [13] employed for the diagnosis of eolian motors. We simulate this model during 1 second and we compute
for each tracked model (via the AT plug-in) the three metrics QA, WA and MCA. The two first ones are computed over the simulation time while the MCC static metric is obtained before the simulation starting.

Figure 3 depicts the 49 ordered models (identified by ID) according to the MCA metric. We can note that there is a significant gap between the 16 models with the highest MCA value (75) and the 9 models with the lowest MCC value (2).

In Figure 4, for each component $i \in D$, normalized activity metrics $NQA_i = \frac{QA_i}{QA_{\text{max}}}$ (with $QA_{\text{max}} = \max\{QA_i \mid i \in D\}$) is compared with normalized CPU metrics $NCPU_i = \frac{CPU_i}{CPU_{\text{max}}}$ (with $CPU_{\text{max}} = \max\{CPU_i \mid i \in D\}$). It can be seen that normalized activity corresponds mostly to normalized CPU.

Figure 5 depicts the distribution of normalized $\frac{NMCA}{NCPU}$ ratio. It can be seen that normalized NMCA constitutes a good prediction for most of components. Correct prediction concerns 27 components over the 49 ones. NMCA overestimates the other components. This means that for some components, NMCA predicts they will have more activity. However, during the simulation, these overestimation could be dynamically corrected.

One can ask: “Why would we still care about correcting the static activity as soon as we have found the dynamic activity, instead of simply using the dynamic activity?” Analytic activity can depend on initial state but also on input event (for firespread a cell can receive water or not impacting its simulation activity). Therefore, one can imagine that a component with a high analytic activity, although the latter at the beginning of the simulation does not exhibit a high level of activity, if it is noticed an increase of activity during the simulation, one can anticipate the component would have a high level of activity and take faster decision/prediction (e.g. for load-balancing in a case of distributed simulation).
5 Perspective: Routed activity

A route $R$ can be defined as the set of port names (including component name to be unic) an output event of a source atomic component crosses to reach a final receiver (another atomic component). Subtracting both original and final ports to the size of $R$ thus corresponds to the number of hierarchy levels crossed by an event. Then, each weighted transition (event) can integrate routing weight, i.e., the number of elements $r_i$ of each route $R_i$ activating a transition type $i$. Each routing weight can be determined at static level.

Input and output sets can be defined as $X = \{(p,v) | p \in P_{in}, v \in V_X\}$, where $P_{in}$ is the set of input ports, and $V_X$ is the set of input values, and $Y = \{(p,v) | p \in P_{out}, v \in V_Y\}$, where $P_{out}$ is the set of output ports, and $V_Y$ is the set of output values. To index external transition types, the notion of ports can be used.

![Figure 5. Distribution of NCPU and NMCA comparison.](image)

Figure 6 presents a hierarchical coupled model $C1$ involving $C2$, $C3$ and $C4$ DEVS coupled models and $A1$, $A2$, $A3$ DEVS atomic models. In Figure 7, the associated dependency graph is used to compute the coupling metric with the following rule: each time a coupled model is traversed by an event, the coupling value is incremented by one (this corresponds to the number of hierarchy levels crossed by an event along a route $R$ as explained before). For example, the coupling metric between $A3$ and $A1$ models is four since the coupling between $A1$ and $A2$ is one.

![Figure 6. Hierarchical coupled model.](image)
6 Conclusions

In this paper, analytic and simulation activity metrics have been presented with a special attention about their relationships. Concerning the analytic activity metric, the new McCabe cyclomatic complexity activity metric \((MCA)\) of a DEVS model are presented. For simulation activity metrics, the well known quantitative-activity metric \((QA)\) has been used in the context of profiling the activity distribution over components. Both \(MCA\) and \(QA\) metrics have been compared with the user \(CPU\) time of the DEVS transition functions are introduced. The \(MCA\), \(QA\) and \(CPU\) metrics have been implemented through a plug-in of the DEVSimPy (DEVS Simulator in Python language) environment and applied to a case study. It is clear that \(MCA\) constitutes a good approximation allowing a-priori estimation of component simulation overheads.

Main perspective concerns the improvement of the simulation algorithm using coupling metrics and associating it with activity. Structure modification of abstract simulators will be guided by a combination of both activity and the coupling metric associated with a parallel and distributed simulation algorithm. DEVSimPy framework already integrates PyPDEVs API [5] thus exploiting parallel and distributed features. We plan also to use an activity prediction model [14] which requires that the user provides (domain-specific) knowledge about how a specific model will use computational resources when simulated with a specific simulator. Futhermore, in order to avoid the break of the model-simulator abstraction (due the fact that the modeller needs some knowledge about the simulator’s operation), we plan to explore the use of Domain-Specific Language (DSL) to automated the construction of the activity prediction model.

References