Modeling and Analysis for Energy-Driven Computing using Statistical Model-Checking

Abdoulaye Gamatié, Gilles Sassatelli
LIRMM, Univ. Montpellier, CNRS, France
first.last@lirmm.fr

Marius Mikučionis
Dep. of Computer Science, Aalborg Univ., Denmark
first@cs.aau.dk

***Abstract*—Energy-driven computing is a recent paradigm that promotes energy harvesting as an alternative solution to conventional power supply systems. A crucial challenge in that context lies in the dimensioning of system resources w.r.t. energy harvesting conditions while meeting some given timing QoS requirements. Existing simulation and debugging tools do not make it possible to clearly address this issue. This paper defines a generic modeling and analysis framework to support the design exploration for energy-driven computing. It uses stochastic hybrid automata and statistical model-checking. It advocates a distributed system design, where heterogeneous nodes integrate computing and harvesting components and support inter-node energy transfer. Through a simple case-study, the paper shows how this framework addresses the aforementioned design challenge in a flexible manner and helps in reducing energy storage requirements.**

***Index Terms*—Stochastic hybrid automata, energy-driven computing, statistical model-checking, energy harvesting and buffering**

I. Introduction

Energy consumption has emerged as a major issue in computing systems in the last decade. The design of current high-performance computing (HPC) systems is facing this energy-efficiency issue, especially for Exascale computing. The high-energy consumption of supercomputers [1] results in increasingly prohibitive electricity bills of HPC infrastructures (several $M/year). On the other hand, modern multiprocessor embedded systems are also concerned by the demand for better energy-efficiency. Various techniques have been applied at different levels, e.g. microarchitecture level (voltage/frequency scaling, clock/power gating, non-volatile memories, accelerators...), and at system-level (heterogeneous architectures, OS runtime scheduling...). Energy-efficiency is very crucial in the mobile and wearable devices for IoT/edge computing. Energy harvesting is therefore regarded as a potential solution in this respect. This gave birth to the *energy-driven computing* paradigm [2].

Energy-driven computing promotes a design concept in which energy harvesting is as fundamental as computing components. Given the frequent intermittent nature of the energy sources, e.g. wind and sun, the goal of the harvesting system is to match the compute system power demand to the available power. There are two complementary ways to reach this goal: adapting the power consumption of the load according to the available energy and introducing energy buffers between power supply and consumption. The literature on energy-driven computing distinguishes different classes of systems [2]. An *energy-neutral* system always maintains the QoS requirements of a load w.r.t. the dynamics of the harvested energy, using energy buffers. Here, an important design goal is to reduce as much as possible the cost of these buffers. A *transient* (or intermittent) system executes its workload whenever enough energy has been buffered and remains inactive otherwise. In that case the system operates in best-effort mode and is therefore unable to guarantee QoS constraints will be met.

Considered problem and solution. We are interested in the following two important design questions: *i) how to design energy-driven computing systems while guaranteeing the timing QoS requirements of the executed applications w.r.t. uncertain and fluctuating energy availability? ii) how to properly size energy buffers so as to minimize both their size and cost (energy storage is a key design component in terms of physical dimensions, product lifetime and therefore cost) under the QoS constraints?* Answering the above questions will help us find the suitable energy-neutrality conditions of the studied systems. Existing simulation and debugging tools dedicated to energy-driven computing do not make it possible to easily answer the above questions [2]. We therefore need a suitable modeling and analysis framework to address this unsolved challenge.

We advocate a modular design framework where an energy-driven computing system is defined by distributed nodes interacting with each other (see Fig. 1). A node is a heterogeneous component that can integrate processing elements, an energy harvester (e.g. solar panel), an energy buffer (e.g. battery), and a local controller managing the node w.r.t. its environment. We consider solar energy as it is reasonably predictable using existing weather forecast databases [3]. While such distributed systems usually support data migration between different nodes, a salient feature of our envisioned system is *inter-node energy transfer*. This is made possible by an unprecedented power crossbar we recently defined [4]. Thanks to this feature, when a node is running out of energy, it can fetch energy from remote nodes to continue processing locally, instead of offloading its computations onto distant nodes. Indeed, a fixed-length period of data migration is an order of magnitude more energy-demanding than a similar period of local computation [5].

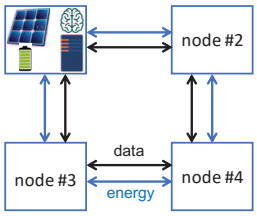


Fig. 1: 4-nodes system

We adopt stochastic hybrid automata and statistical model-checking (SMC) [6] to implement our framework. Such au-

tomata generalize timed automata by using continuous variables in place of clocks. In addition, clocks can have various rates, specified by either constants or expressions on variables in the form of *ordinary differential equations*. In SMC, properties are verified on randomly generated and monitored simulation runs of a system. Then, statistical analysis is applied to compute an estimate of the probability for a model to satisfy some properties, up to a certain confidence degree. While traditional symbolic model-checking enables to verify properties with 100% confidence, this is usually very expensive (even unfeasible) for huge systems. SMC overcomes this scalability issue by bounding the probability of making an error in verification via the number of simulation runs.

Our contribution. We define the main components of energy-driven computing systems in the well-known Uppaal tool: applications, execution platforms, energy harvesting, and buffering. The components are described by automata that can be easily instantiated and extended. They are analyzable with Uppaal SMC. Through a simple yet relevant case-study, we address the energy storage minimization in a distributed system that executes a real-time application under deadline constraints. Then, we show how energy migration in such a system helps sizing the energy buffers for reducing their overall cost.

II. Related work

Albeit the high demand for simulation and debugging frameworks in energy-driven computing, only a few candidate studies exist [2]. In [7] [8] authors focus on micro-scale systems (RF and wireless sensor nodes) and address the erratic dynamics of harvesting sources. They leverage current-voltage traces, generated from real energy harvesters, to reproduce and predict realistic harvesting conditions for testing in the lab. The Ekho emulator [7] fulfills this demand and favors repeatable experiments. The energy-neutrality of a system is achieved by applying workload management techniques [8]. None of the two approaches provides fine-grain introspection capabilities of system behaviors, which matters much for performing a proper sizing of the energy subsystems (harvester, energy buffers etc.).

In [9], a system prototyping flow based on a simulator of non-volatile processor is proposed for IoT. It takes as input the power traces of the harvester and the execution platform characteristics to analyze the temporal and energy behavior of a system. The authors studied the impact of different capacitor sizes. While their approach is relevant, there is no attempt to model a realistic energy harvester which also has an important impact on the proposed analysis flow. The CleanCut approach [10] adopts a software-oriented design for preventing program state corruption in energy harvesting devices. Energy harvesters are not explicitly addressed here, contrarily to energy storage.

Unlike the above studies, we here advocate a high-level design-time approach for holistic modeling and evaluation of energy-driven multiprocessor systems. The resulting framework is flexible enough to model all features of the target systems. Thanks to SMC, one can explore various system properties with an acceptable level of confidence. A previous work [11] already applied Uppaal SMC to battery-aware task scheduling in satellite systems. The authors modeled a kinetic battery as a stochastic hybrid system and study its performance. The present work differs from [11] by proposing a system-level approach for designing distributed systems supporting energy transfer.

III. Framework for energy-neutral systems

Our framework consists of parameterized automata templates associated with the system components. An automaton consists of states and transitions. A state can be associated with invariant properties and user-defined rates (exponential probability distributions). Transitions between states can be associated with Boolean conditions, synchronizations via a channel `c` (using the primitives `c?` and `c!`), and updates of variables and clocks. For the obvious sake of concision here we will merely illustrate these notions through the subsequent system modeling.

A. Computing components

Our computing system modeling is inspired by the scheduling framework demo example provided in Uppaal [12]. We borrow three concepts from this framework: *task, resource* (i.e. processing element) and *scheduling policy*. However, we extend the former two concepts to reason on energy, beyond temporal properties. The static parameters `tid` and `rid` are respectively used to distinguish different task and resource instances.

The considered task model is shown in Fig. 2, with one such an automaton instance per task. It originally captures periodic, sporadic or aperiodic task execution and takes into account inter-task dependencies. These can be set via the task graph attributes in the Uppaal model [12]. We extend this model to manage a task when its executing resource is running out of energy. In this case, the task is suspended and enters a `Frozen` state (see Fig. 2) until energy becomes available which then resumes task execution. In case its deadline is missed, the task goes to an `Error` state. Note the use of a stopwatch in Uppaal modeling, via the invariant expression `x'==0` to temporarily freeze the execution time of the task. When the task returns to the `Ready` state (i.e. this state of the automata denotes either a task is being ready for execution or the task is running), the time progress is resumed via the invariant `x'==isRunning()`. The `Ready` state has a rate of exponential of 10^6, which determines the probability of leaving the state in SMC (the higher the rate, the sooner the state is left whenever possible).

Beyond the task model, we also extend the resource model (Fig. 3), to account for its dissipated power during task execution. From the initial state `Idle`, a ready task is either inserted in a waiting queue when the resource is already occupied by other tasks (denoted by `!empty()`), or directly executed. On completion, the resource executes the remaining tasks if any, or returns to `Idle` state. In this paper, we introduce a cost variable `dissipated_power[rid]`, characterized by the resource identifier `rid`. Power dissipation occurs when the resource automaton is in the `InUse` state, meaning a task is being executed (among those assigned to this resource). For the sake of simplicity, we consider a cost formula consisting of the product of three terms. The term `pow_coeff[rid]` characterizes the microarchitecture complexity of the resource, which has a direct impact on both dynamic and static power. Typically, an application-class processor will dissipate more power than

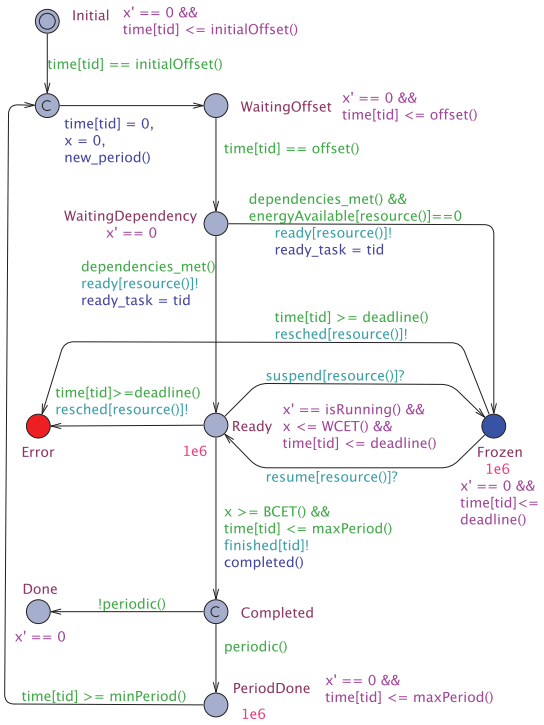


Fig. 2: Task template

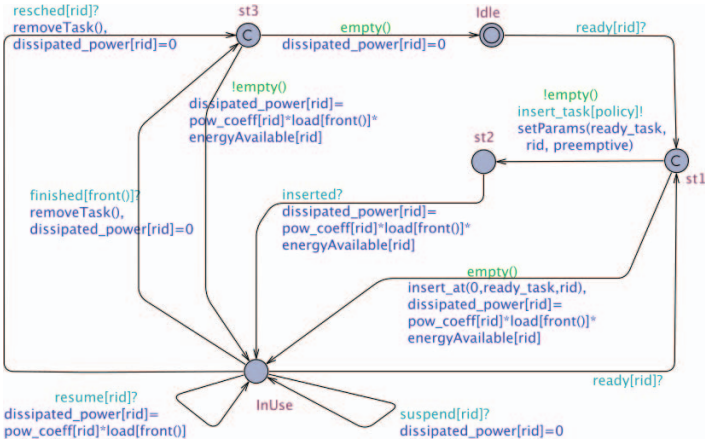


Fig. 3: Execution resource template

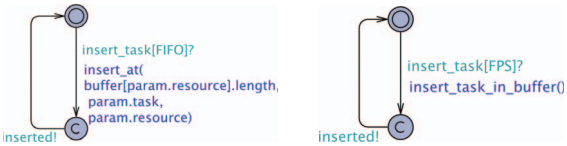


Fig. 4: FIFO (left) and Fixed-Priority (right) scheduling policies

a microcontroller since its microarchitecture is more complex, e.g. in terms of pipeline depth, branch prediction. Both homogeneous and heterogeneous multiprocessor platforms can be modeled via different values for `pow_coeff[rid]`. The term `load[front()]` denotes the workload intensity of the running task. The identifier of this task is obtained via the *front()* function defined locally in the resource template. The term `energyAvailable[rid]` is equal to `1` when enough energy is available for processing, `0` otherwise.

Finally, we use the task scheduling policies modeled in [12]. They are invoked on the transition between the states `st1` and `st2` (see in Fig. 3). In Fig. 4, the left-hand side policy inserts tasks in waiting queues according to the First-In-First-Out order while the other one exploits task priorities. Here, `buffer[..]` is used as a task waiting queue of each resource.

B. Harvesting system components

A solar panel model is shown in Fig. 5. It outputs power according to a local function defining daily irradiance conditions, invoked on the self-loop transition, as follows:

```
double solar_panel(double gTime) {
   const double PI=3.14159265; double totalPow, result;
   double timeOfTheDay = fmod(gTime,DAY);
   irrad = HEIGHT * ((1/(SIGMA * sqrt(2.0*PI))) *
      exp(-((timeOfTheDay + OFFSET - MU) *
            (timeOfTheDay + OFFSET - MU)) / (2*SIGMA*SIGMA)));
   return irrad * panel_area * conv_eff * loss_coeff;
}
```

The variable `timeOfTheDay` indicates the time in minutes during a day, hence the *fmod* modulo function applied to the input parameter `gTime`, a global increasing clock. The value of the constant `DAY` is `1440`, i.e. number of minutes in a full day. Then, we express the daily solar irradiance, `irrad`, by using the probability density function of the Gaussian distribution (the constants `MU` and `SIGMA` are the mean and standard deviation of the distribution), multiplied by the constant `HEIGHT` to upscale the peak value of the curve above [0,1]. To capture different sunrise scenarios, we introduced the `OFFSET` constant. Then, the solar panel output is given by the product of `irrad`, `panel_area`, the panel conversion efficiency `conv_eff`, and a loss coefficient `loss_coeff` related to the power transfer from the panel to energy storage.

The energy storage mechanism is modeled via a rechargeable battery manager shown in Fig. 6. The static parameter `bid` identifies each manager instance. Each battery is initially assumed to be full. In the corresponding state, we use an array of hybrid clocks `battery_energy` indicating the current energy budget. The invariant `battery_energy[id]'==0` freezes the evolution of each clock in `Full` and `Empty` states. To estimate the amount of time during which a battery has been empty, we use a Stopwatch via the clock array `depleted`. Only the invariant of the `Empty` state allows such clocks to progress, noted `depleted[id]'==1`. When the energy of a battery decreases, it goes to the `Nominal` state. The associated invariant for `battery_energy[id]` is a derivative equation which therefore integrates power (solar panel power minus dissipated power) over time for keeping track of battery level. When the energy in the battery is depleted, the automaton switches to the `Empty`

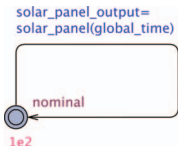


Fig. 5: Solar Panel

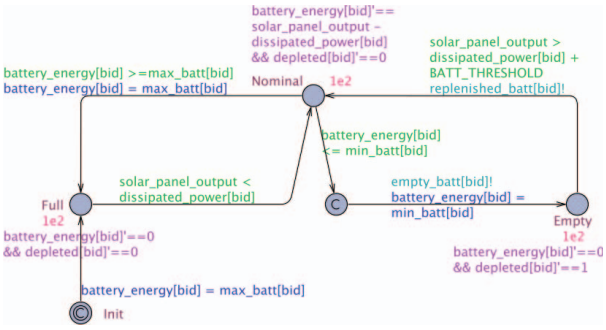


Fig. 6: Battery manager

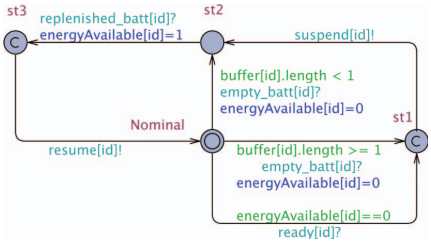


Fig. 7: Energy-driven system controller

state. It will return in `Nominal` above a user-defined power threshold for recharging.

To orchestrate the interaction between the computing and harvesting components, we devise a bridge called the energy-driven system controller in Fig. 7. It is associated with each system node and characterized by the static parameter `id`. Whenever a battery becomes empty (denoted by the `empty_batt[id]` channel emitted by its manager), the associated execution resource and task are suspended. This is expressed via the `suspend[id]` channel sent to the task and resource automata. Whenever the battery manager emits a synchronization request on the `replenished_batt[id]` channel, the controller resumes the task execution on its assigned resource, modeled via the `resume[id]` channel.

IV. Application of the Framework

A. A first design evaluation

We address the problem of minimizing the energy storage required for sustaining energy-neutral execution under deadline constraints. We illustrate the modeling on an application graph inspired by the Rosace case-study [13]. While this real-time application originally operates in a few seconds, we deliberately modified its time scale to the range of minutes (see Tab. I). This enables to analyze relevant execution scenarios spread over a whole day, under different irradiance conditions.

TABLE I: Rosace application (**bcet**, **wcet** and **prio** respectively denote the best/worst-case execution times and task priority).

tasks	init offset	min/max periods, deadline	offset	bcet, wcet	resource	prio	load
T0	0	120	0	5	0	1	6
T1	0	120	0	5	1	1	6
T2	0	120	0	10	1	1	13
T3	0	240	0	5	0	1	6
T4	0	240	0	5	1	1	6
T5	0	240	0	5	2	1	6
T6	0	240	0	5	3	1	6
T7	0	240	0	5	4	1	6
T8	0	480	0	5	0	1	10
T9	0	480	0	5	1	1	10
T10	0	480	0	5	4	1	10

We instantiate five nodes, each comprising an execution resource, a solar panel, a battery manager and an energy-driven controller, resulting in as many automata instances in the system. Each execution resource features an Odroid-XU compute board [14], with a maximum power dissipation of 13W. We consider a solar panel area of 4 dm^2 with a conversion efficiency of 0.23. In each energy transfer transaction, a loss of 5% is applied. As for harvesting conditions, we consider clear-sky daily irradiances in the mediterranean city Girona in Spain, during June and December, with a maximum peak values of 973 and 423 W/m^2 respectively. They are obtained from a public database [3], developed by the European Commission's science and knowledge service for open research.

To address the problem of interest, we first model a conventional system setup where energy is always available (i.e. grid-connected system). This is obtained by initializing all elements of the array `energyAvailable` to `1`, and by composing the instances of the automata shown in Figs. 2 – 4. Then, we determine application mappings and schedules on the multi-processor platform, which violate no deadline. Tab. I shows a task/resource mapping that meets this requirement.

Now, let us consider the same system, but connected to a calibrated harvesting system, i.e. augmented with instances of the components models shown in Fig. 5 – 7. The energy availability of the system depends on the irradiance conditions.

We apply Uppaal-SMC to empirically explore candidate battery sizes for a 2-days execution of the system. In practice we instantiate the system with different battery sizes, and we repeatedly check the following query of the model-checker:

```
Pr[<= 2*DAY] (<> exists(i:t_id) Task(i).Error)
```

This query estimates the probability that any task goes to an `Error` state, i.e. deadline violation. It enables us to quickly identify admissible battery sizes with a first confidence interval, through about 29 simulation runs, in less than 2 minutes (on a desktop computer including an Intel i7-8700 CPU at 3.20GHz with 31.2GB memory).

Once a reduced set of candidate battery sizes is identified, we confirm the statistical relevance of each size value. We then check the model again for a very high number of random simulation runs. For this purpose, we use the following query:

```
simulate [<=2*DAY; 1000] {...} : 1 : exists(i:t_id)
    Task(i).Error
```

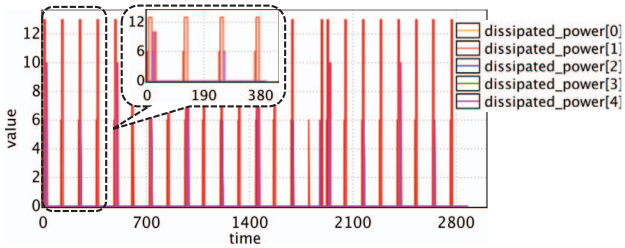
It checks whether any `Task` instance goes into an `Error` state, for 1000 random simulation runs over two full days. The dots appearing between the brace brackets can be filled with

some variables of the system model to monitor their evolution towards a state satisfying the checking query. This is useful for the model debugging. The evaluation time of each new query instance takes about one hour. This justifies the exploration space reduction using the first query. Tab. II summarizes the minimum battery sizes in Watt-hours (Wh) obtained for 5 system nodes, which guarantee energy-neutrality and real-time constraints in June and December. The total battery size in case of homogeneous battery system is also given in the last column, i.e. system with identical batteries of 23.4Wh for each node.

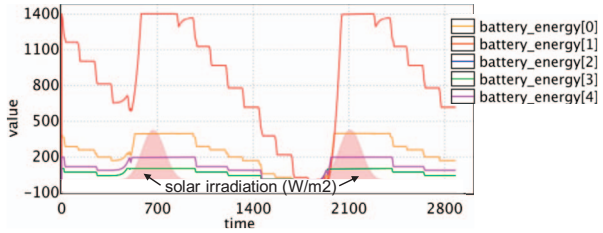
TABLE II: Min. battery size (Wh) to meet deadline constraints

	B0	B1	B2	B3	B4	Total	Total homogen.
June	4.7	16.8	1	1	1.7	25.2	84
December	6.7	23.4	1.7	1.7	3.3	36.8	117

Fig. 8 illustrates a 2-days system execution trace in December (Girona-D) with the corresponding battery sizing (time scale is in minutes). Batteries' evolution (i.e. the curves) is shown together with the periodic power dissipation from the resources/processors executing tasks. We observe that all batteries can temporarily be empty in the night between the two days, while still sustaining the system energy-neutrality.



(a) Power dissipated by periodic load (in Watt)



(b) Battery energy evolution (in Watt-minutes)

Fig. 8: 2-days execution scenario (Girona-D)

B. Alternative design: integration of energy transfer

We explore a new system design aiming at reducing the cost of the batteries under energy-neutrality and real-time constraints. We extend the previous framework with energy transfer capability between interconnected nodes, such that we turn the battery array into a distributed energy pool: energy stored in these batteries is mutualized across the whole system. Another expected advantage is the system resilience improvement through the integration of redundant nodes equipped with batteries. Typically, a node with battery failure could remain operational by fetching energy from remote batteries.

To support energy transfer, we only extend two components: the battery manager and the energy-driven system controller. In Fig. 9, the battery manager features energy fetching from remote batteries when a local battery is empty, expressed via a new channel `fetched_energy[id]`. In addition, when the local battery gets empty upon an energy transfer towards remote nodes, it is notified via the new channel `notify[id]`. In Fig. 10, the energy transfer is addressed by the bottom part of the extended controller. It relies on a simple strategy here: when a the local battery of a node contains more than 50% of its total capacity, a remote node requesting energy transfer (through its system controller) may fetch 50% of the local battery; otherwise, all of the energy in the local battery is transferred. The node providing remote nodes with energy is randomly selected. A loss coefficient is applied to each transfer.

We configure and evaluate new designs including more than 5 energy-sharing nodes. These are equipped with smaller batteries such that their global capacity does not exceed the setup found previously in Tab. II. The mapping of application tasks is unchanged, i.e. 5 nodes are used for task execution. The other nodes therefore only act as energy harvesters/storage units making their energy available to the first 5 nodes.

Considering the same exploration process as for the initial design, we focus on the battery size minimization problem. After exploring different battery configurations for 2-days execution in December in Girona, the selected solutions are reported in Tab. III and include 2 heterogeneous and one homogeneous configurations. For both *heterogeneous* configurations we observe that the total energy required is marginally higher than previously (due to energy transfer losses). Note however the third *homogeneous* configuration which requires over 65% less energy than previously (see Table II). This shows a straightforward homogeneous design with one single conventional 18650 battery (7Wh) would here prove enough.

TABLE III: Min. battery size (Wh) for 6 & 8 nodes in Girona-D

# Nodes	B0	B1	B2	B3	B4	B5	B6	B7	Total.
6 (het.)	5.8	10	5.8	5.8	6.8	4.2	–	–	38.4
8 (het.)	5	6.8	4.2	4.2	5.5	4.2	4.2	4.3	38.4
8 (hom.)	5	5	5	5	5	5	5	5	40

Fig. 11 illustrates a valid 2-days execution trace of 8-nodes system, with energy transfer. We can observe the multiple energy migrations that occur between the mutualized batteries.

Beyond the cost consideration regarding the batteries, it is interesting to check whether they highly differ w.r.t. their emptiness duration for a 2-days execution. We use the following Uppaal-SMC query to estimate the maximum time during which the energy is depleted in each battery:

```
E[<=2*DAY; 1000] (max: depleted[...])
```

It estimates the maximum value of each clock component of the array `depleted` (used in the battery manager model), across 1000 random simulation runs over 2-days. The computed delays in minutes for the Girona-D scenario are reported in Tab. IV. In both configurations, the batteries are empty between 4% and 9% of the time corresponding to the 2-days execution.

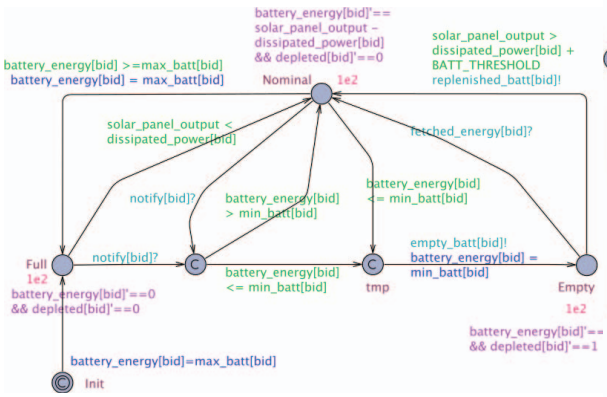


Fig. 9: Battery manager for energy sharing

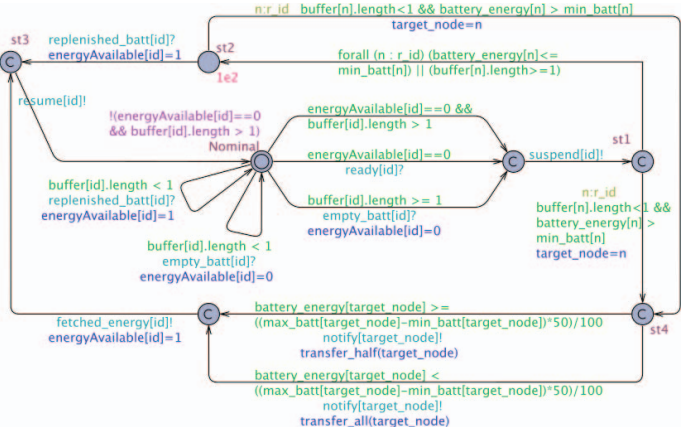


Fig. 10: Energy-driven system controller for energy sharing

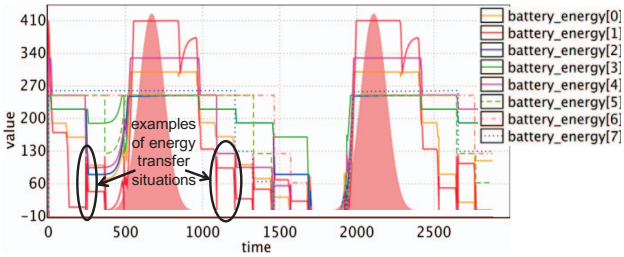


Fig. 11: Battery state (with transfer) over 2 days for Girona-D

TABLE IV: Percent. of time with empty batteries for Girona-D

# Nodes	B0	B1	B2	B3	B4	B5	B6	B7
6 (het.)	10.3	6.1	9	8.9	8.6	10.9	–	–
8 (het.)	9.7	6.2	10.4	10.2	10.3	9.7	9.6	9.7

V. Conclusion and Perspectives

In this paper, we proposed a high-level modeling and analysis framework devoted to energy-driven computing systems. We used stochastic hybrid automata and SMC, which makes it possible to quickly assess the energy-neutrality property of candidate system configurations and therefore helps for hardware component sizing. The case-study chosen for this paper entails advanced features such as the innovative distributed energy pooling technique with local decision making which could be properly modeled and assessed with this framework. The automata templates defined in the framework of this work are made available in the Uppaal Model Repository [15].

Beyond the obvious refinement of the many physical models (e.g. weather models, idle node power consumption, battery wear), future work will rely on elaborating and assessing smarter energy migration strategies. The guarantee of energy-neutrality and QoS requirements could also rely on the co-optimization of both the computing and energy subsystems.

References

[1] Wiki., "Supercomp." 2020, https://en.wikipedia.org/wiki/Supercomputer.
[2] S. T. Sliper, O. Cetinkaya, A. Weddell, B. Al-Hashimi, and G. Merrett, "Energy-driven computing," *Philosophical Transactions of the Royal Society A: Mathematica, Physical and Engineering Sciences*, vol. 378, no. 2164, pp. 1–4, Feb 2020.
[3] European Commission's science and knowledge service for open research, "PVGIS Tools," 2020, https://ec.europa.eu/jrc/en/pvgis.
[4] F. Di Gregorio, G. Sassatelli, A. Gamatié, and A. Castelltort, "A Flexible Power Crossbar-based Architecture for Software-Defined Power Domains," in *22nd European Conference on Power Electronics and Applications (EPE'20 ECCE Europe)*, Lyon, France, Sep. 2020.
[5] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent Computing: Challenges and Opportunities," in *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, ser. Leibniz Int'l Proc. in Informatics (LIPIcs), vol. 71, Dagstuhl, Germany, 2017, pp. 8:1–8:14.
[6] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen, "Uppaal SMC tutorial," *Int. J. Softw. Tools Technol. Transf.*, vol. 17, no. 4, pp. 397–415, 2015.
[7] J. D. Hester, T. Scott, and J. Sorber, "Ekho: realistic and repeatable experimentation for tiny energy-harvesting sensors," in *12th ACM Conf. on Embedded Network Sensor Systems, SenSys'14, Memphis, Tennessee, USA*, Á. Lédeczi, P. Dutta, and C. Lu, Eds., 2014, pp. 330–331.
[8] A. Savanth, A. Weddell, J. Myers, D. Flynn, and B. Al-Hashimi, "Photovoltaic cells for micro-scale wireless sensor nodes: measurement and modeling to assist system design," in *3rd Int'l Workshop on Energy Neutral Sensing Systems (ENSsys 2015)*, September 2015.
[9] Y. Wu, Y. Sun, Z. Jia, L. Zhang, Y. Liu, and J. Hu, "Prototyping energy harvesting powered systems with nonvolatile processor (invited paper)," in *2018 Int'l Symp. on Rapid System Prototyping (RSP)*, 2018, pp. 49–55.
[10] A. Colin and B. Lucia, "Termination checking and task decomposition for task-based intermittent programs," in *27th International Conference on Compiler Construction, CC 2018, Vienna, Austria*, 2018, pp. 116–127.
[11] Erik R. Wognsen, Rene R. Hansen, and Kim G. Larsen, "Battery-aware scheduling of mixed criticality systems," in *Int'l Symp. On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14*. Springer, 2014, p. 208–222.
[12] Uppaal team, "Demo example - SchedulingFramework.xml," 2020, http://www.uppaal.org.
[13] F. Boniol, Y. Bouchebaba, J. Brunel, K. Delmas, T. Loquen, A. Mascarenas Gonzalez, C. Pagetti, T. Polacsek, and N. Sensfelder, "PHYLOG certification methodology: a sane way to embed multi-core processors," in *Cong. on Emb. Real-Time Soft. and Sys. (ERTS)*, Toulouse, Fr., 2020.
[14] Hardkernel, "Odroid-XU big.LITTLE board," 2020, https://www.hardkernel.com/shop/odroid-xu/.
[15] Uppaal team, "Uppaal Model Repository," 2020, https://deis-tools.github.io/uppaal-models/CaseStudies/EnergyNeutrality.