Energy Management with TRIANA on FPAI

Hermen A. Toersche, Computer Architecture
Johann L. Hurink, Discrete Mathematics
University of Twente
Enschede, the Netherlands
{h.a.toersche, j.l.hurink}@utwente.nl

Mente J. Konsman
Business Information Services
TNO
Groningen, the Netherlands
mente.konsman@tno.nl

***Abstract*—The current growth of smart grid capable appliances motivates the development of general and flexible software systems to support these devices. The FlexiblePower Application Infrastructure (FPAI) is such a system, which classifies devices by their type of flexibility. Subsequently, energy applications only have to support these flexibility classes. In this work, we present an implementation of the TRIANA demand side management approach as an energy application on the FPAI energy management software platform. We use dynamic programming to solve the local scheduling problems for each flexibility class. This work shows that FPAI can host energy applications with different control approaches and that the TRIANA control approach can be embedded in a general implementation framework.**

***Index Terms*—Energy management, middleware, dynamic programming, smart grids.**

I. Introduction

The introduction rate of “smart grid ready” devices on the market has come to a point where developing a specific management approach for every specific device type is unsustainable. As a consequence, the developers of demand side management (DSM) software and devices should cooperate and introduce a standard for device flexibility.

The most straightforward approach, which is also proposed in current standardization efforts (e.g. [1]), is to steer devices by sending prices to devices. While this approach may solve the problem from the perspective of the DSM software to some extent, the burden of scheduling the device is left to the manufacturer. However, developing a good scheduling algorithm takes time and experience. Under time-to-market pressure, it is very likely that a developer puts in only minimal effort to “support” a standard. Furthermore, due to differences between the proposed standards, it may even be necessary to develop multiple front-ends or multiple scheduling implementations to target different standards. As a consequence, there is a need to separate the device control problem from the device scheduling problem.

The FlexiblePower Application Infrastructure (FPAI) platform [2] addresses this gap between DSM software and device drivers. Drivers do not solve optimization problems directly, but instead provide a description of the flexibility that a device offers to the platform. In turn, an energy application (implemented by the DSM software) exploits this flexibility. The driver translates the allocation to device control actions.

The FPAI platform defines a set of very general energy flexibility classes (*control spaces*). To limit the effort to develop and maintain energy applications, the set of classes is small. Many details of the environment (e.g. the market mechanism) are only exposed to the energy applications and not to the device drivers. As a result, the platform is very flexible.

This paper introduces an FPAI energy application that implements the TRIANA DSM approach [3]. We briefly present TRIANA, and the context that leads to the FPAI platform, in section II. Next, we focus on the challenges to port TRIANA to this platform, and propose solutions to overcome these challenges in section III. We demonstrate and evaluate the implementation with simulation experiments in section IV.

II. Background

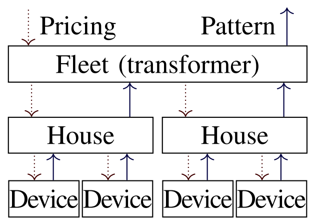
A. Home Automation And Energy Management

Demand side management depends on control over the demand, i.e. over devices. Home automation is a popular field which needs this control as well, but mostly focuses on aspects unrelated to energy. A large array of standards and software platforms have emerged for home automation, e.g. [4] and [5].

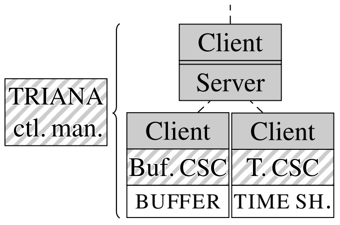
Independently, standards have developed for energy management, e.g. OpenADR [1]. Several recent home automation standards incorporate energy control features as well [6], [7]. These standards address energy flexibility, rather than user-facing functionality. Residential standards focus on demand response with heating, ventilation, and air conditioning (HVAC) systems, because these represent the bulk of the controllable load, especially in the US. In larger buildings, building energy management systems are common, which also focus on HVAC. These management systems have (vendor-)specific optimization solutions, e.g. [8]. Support for integration with other (in-building) systems is usually limited or not available.

During the development of the PowerMatcher DSM approach [9], it was found that the development of a control agent for every possible device type is infeasible. As an alternative, FPAI proposes that device drivers expose the *structure* of the energy flexibility. Subsequently, structurally similar problems can be addressed with a single agent. The effort in agent development has been acknowledged in standards development by providing versions with limited features [10]. We believe that FPAI avoids this effort without compromising functionality.

This work is funded by EIT ICT Labs as part of the HEGRID project (#13114). Part of this research is conducted within the DREAM project supported by STW (#11842).



(a) Hierarchical control in TRIANA.



(b) House level control on FPAI.

Figure 1. Partitioned optimization approach in TRIANA.

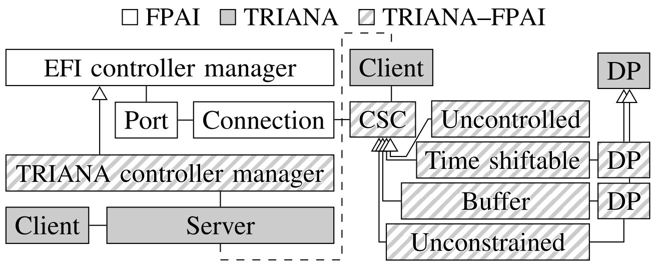


Figure 2. Overview of TRIANA on FPAI software architecture, with components separated by category: FPAI, TRIANA, and the TRIANA–FPAI binding.

B. *TRIANA Demand Side Management*

TRIANA is a planning based approach for large scale distributed demand side management of households in smart grids [3]. The approach can be used for numerous demand side management applications, ranging from the operation of a fleet of microCHP generators to planning electric vehicles (EVs). TRIANA accounts for both the global and the local control problems in a system. The predictive control process is divided in three stages: forecasting, planning and operational control. For scalability, the problem is partitioned along its hierarchical structure, e.g. on a device level, house level, and transformer level (see Figure 1a). A hierarchical feedback process iteratively refines the behavior of a fleet of devices.

C. *FPAI Energy Management Platform*

FPAI defines a message protocol and a reference implementation for the management and control of smart appliances. FPAI is implemented in Java on top of OSGi. The implementation of FPAI is open source and available online [11].

FPAI offers the following control space classes:

- UNCONTROLLED describes devices that offer almost no flexibility, for example a photovoltaic (PV) inverter. Device drivers can only selectively allow curtailment.
- TIME SHIFTABLE describes devices for which a “job” has to be executed with start times and deadlines, e.g. a washing machine. Jobs can have multiple parts with a known demand profile and time limits between parts.
- BUFFER describes devices which can be characterized by a continuous state variable (i.e. a state of charge (SoC) or fill level), a set of actuators, losses and demands. An actuator can have multiple modes. A mode describes the behavior of the actuator as a piecewise constant, partial function of the state variable. System descriptions can define a minimum time distance between certain mode changes. FPAI describes these time constraints with timers. A leakage function describes loss over time. Prominent examples of BUFFERs are batteries and thermal systems with storage.
- UNCONSTRAINED devices are in principle BUFFERs without the continuous state variable, for example a diesel generator. These devices have fewer restrictions than BUFFER devices, and thereby allow specific optimizations.

Some devices may fit to multiple control space classes (e.g. EVs). In that case, the FPAI device driver developer should pick the class he assumes to be most appropriate.

III. TRIANA ON FPAI

The TRIANA port for FPAI is part of a software system. Therefore, we first give a brief overview of the components of TRIANA in the context of the FPAI platform. The second part of the section focuses on optimization techniques for the device flexibility classes.

A. *Software Architecture*

Figure 2 presents an overview of how TRIANA can be implemented on the FPAI platform. We separate the components in three categories: FPAI, the TRIANA core library, and the binding of TRIANA to FPAI. FPAI and TRIANA are independent, sharing only the Java/OSGi runtime. The TRIANA–FPAI binding connects TRIANA and FPAI.

B. *TRIANA Core Library*

TRIANA sets up a control hierarchy using the TRIANA core library. The core library implements the communication and protocol handling between the different entities. The library is split up in a client part and a server part, with several auxiliary services. The client part is responsible for interfacing with an aggregator. The server part is responsible for local aggregation.

The `Client` communicates with an aggregator. The client gives `Pricing` descriptions to the connected resource (i.e. device, house, etc.). In response, the resource should return a demand `Pattern`, which proposes a consumption plan over time. The aggregator selects the most useful pattern according to the desired group behavior. If the aggregator is unavailable (as for the root node, or when the service is unreachable), the client locally takes over this role.

The `Server` fulfills a local aggregator role. It manages groups of clients (i.e. devices), and embeds algorithms for the coordinated scheduling of groups. In the current implementation, only the IDDP group scheduling algorithm [3] has been ported to Java, and adapted to support multi-commodity optimization. However, we see no technical limits which prevent porting algorithms from other work. A server contains a `Client`, which enables to set up a hierarchical control for groups of devices, where a server acts as a sub-aggregator. The first aggregation step is typically at household level, with further aggregation steps upstream (as in Figure 1a and 1b).

As an example, a house controller (server) asks the devices (clients) for a set of candidate profiles, and chooses a combination of these profiles which together give a good profile for the aggregator and the local objective (see Figure 1b).

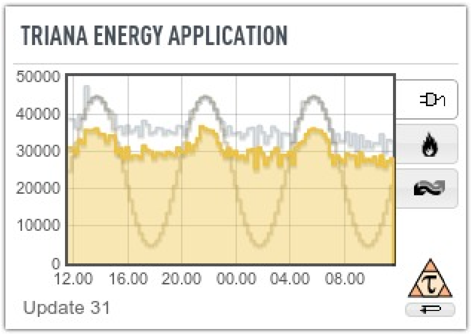


Figure 3. Screenshot of TRIANA–FPAI energy application widget, which shows the day-ahead electricity demand planning for a simulation of 50 microCHPs and 50 heat pumps on top of a sinusoidal baseload profile. Note that the profile is not completely flat due to multi-commodity optimization, which also aims at reducing peaks in gas demand (not shown in figure).

C. Controller Manager

The central component in TRIANA on FPAI is a "controller manager", which interfaces with FPAI and embeds a TRIANA server. The manager handles dynamic device registration, and provides an FPAI widget graphical user interface for the TRIANA server. Figure 3 shows a screenshot of this widget. While the controller manager is presented as an "energy application", it only acts as glue logic between FPAI, TRIANA, and the TRIANA–FPAI specific user interface.

D. Control Space Controllers

Analogous to the concept of device drivers in operating systems, energy applications in FPAI should provide a driver for each of the control space classes. To distinguish between device drivers and energy application drivers, we refer to these energy application drivers as control space controllers (CSCs). The controller manager instantiates a CSC when a device connects.

A CSC presents a controllable resource as a TRIANA client. The TIME SHIFTABLE and BUFFER CSCs use a specialized dynamic programming (DP) formulation (see [12] for background on DP). UNCONTROLLED uses a greedy algorithm, and UNCONSTRAINED shares the DP with the BUFFER class.

In the following, we present a corresponding scheduling approach for each of the control space classes. We dedicate extra attention to the BUFFER class, because this class is less straightforward.

1) Uncontrolled: The UNCONTROLLED control space describes permissible curtailment ranges over time. In return, the energy application configures the curtailment target range. The corresponding optimization problem has no state. As a result, the problem degenerates to a greedy local selection of the minimum cost. For most pricing structures (e.g. linear, quadratic, ...), this selection is trivial; for linear pricing it means picking the minimum or the maximum permitted curtailment value, depending on whether the price is positive or negative. For notation consistency, we formally consider it as a degenerate DP, with a state space $\langle t \rangle$, where $t \in \{1, \ldots, n_t\}$ is the time interval.

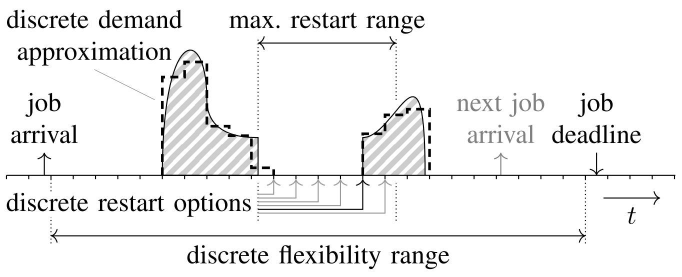


Figure 4. Structure of TIME SHIFTABLE control space discretization for DP.

2) Time Shiftable: The TIME SHIFTABLE control space describes a sequence of static demand patterns (segments), with constraints on the start times at which these patterns are started. We represent states with a pair $\langle t, s \rangle$, where $t \in \{1, \ldots, n_t\}$ is the time interval and s is the number of finished segments.

The planning has to account for the maximum off-time between segments. This accounting may be implemented by adding a timer variable to the state. However, this adds complexity and increases the size of the state space. Instead, we avoid representing these timer variables by exploring all possible off-time values that are feasible in the current state. For each value, we let the next segment start immediately, except at the end of a job. Figure 4 illustrates the discretization of the TIME SHIFTABLE control space for the DP.

3) Buffer: The BUFFER control space describes the response of a dynamical system in a given state (running mode and SoC), subject to the behavior of a group of actuators, buffer demand and losses. Actuators have discrete modes, which describe the behavior of the actuator with a piecewise constant function. The behavior description includes the fill level change over time and the energy commodity demand. There are restrictions on switches between modes. The model is similar to the timed automata formalism [13]. In response to the control space, the energy application returns a schedule of mode switches for each actuator.

To schedule BUFFERs, we use a model-based value iteration DP approach [12]. We only present the details that are specific to the problem at hand. We represent states with a tuple $\langle t, m, s, [\mathsf{T}_1, \ldots, \mathsf{T}_{n_\mathsf{T}}] \rangle$, where $t \in \{1, \ldots, n_t\}$ is the time interval, m is the mode, s is the discretized fill level and T_i is the transition timer state (counting down to 0) for $i \in \{1, \ldots, n_\mathsf{T}\}$, where n_T is the number of timers. We keep the time interval implicit using n_t maps, indexed by t.

Transition timers can, in the worst case, lead to a combinatorial growth of the state space. We abstract timers to a discrete time Markov chain (DTMC) model: rather than representing the explicit timer countdown, we model the timer expiration as a memoryless random process with an average timeout equal to the timer duration. After each time interval, there is a probability of τ/T_i that timer i expires, for $i \in \{1, \ldots, n_\mathsf{T}\}$. The probability is represented with weights in the DP evaluation. The DTMC abstraction reduces the state space of each timer to two states, i.e. *active* and *inactive*.

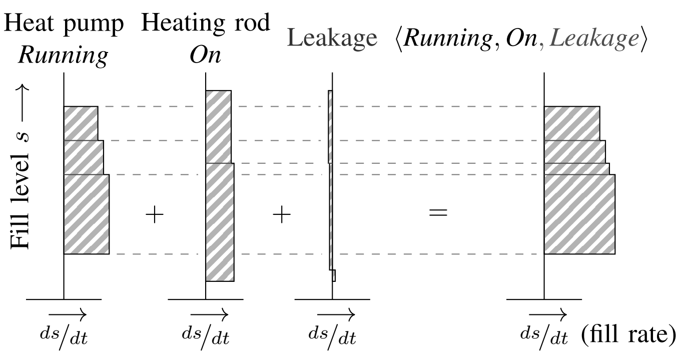
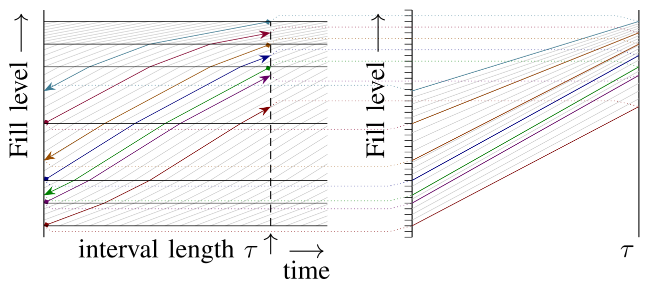


Figure 5. Example of BUFFER fill level function merging with two actuators and storage leakage. The functions map a fill level s (vertical) to a fill rate ds/dt (horizontal). The right-hand function represents the sum of the fill rate functions to the left.



(a) Fill level dynamics for one mode. The horizontal lines represent piecewise break points. The faded diagonal lines give projections of fill rate over time, crossing multiple segments. The arrows give these projections for the break points at $t = 0$ (forward) and $t = \tau$ (backward).

(b) Preprocessed fill level dynamics, for the same value of τ. The ticks on the left hand axis represent the subsequent discretization step, which evaluates the function for specific fill level values.

Figure 6. BUFFER preprocessing example for a "charging" running mode.

A BUFFER has a large, multidimensional state space. To keep the DP tractable, approximations are needed to limit the size of the state space. Furthermore, the evaluation of the state space must be efficient. To these ends, we precompute an internal simplified representation of the model. To limit the effort spent on preprocessing, we exploit that the system dynamics and mode transition model are time invariant. Therefore, we may reuse this representation between planning steps as long as the system description does not change.

a) Preprocessing: The first preprocessing step simplifies the problem by expanding the Cartesian product of the actuator automata. Note that the product expands only the states, and not the combinations of transitions. To incorporate the leakage function, we translate it to an actuator with a single state.

Each of the states in this Cartesian product represents a set of concurrent modes, one for each actuator. To merge the piecewise constant functions of these modes, we use an event sort/merge algorithm. For each of the modes, the algorithm records the steps between the piecewise function segments. A heap implementation is used to order the events by fill level. The resulting algorithm runs in $\mathcal{O}(n_s \log n_s)$, where n_s is the number of pieces in the modes at hand. For brevity, we choose not to reproduce the algorithm in this paper.

Figure 5 illustrates an example of merging a fill rate function. The example combines the *Running* mode of a heat pump with the *On* mode of an auxiliary heating rod (plus storage leakage over time). The figure gives the fill rate (ds/dt, horizontal) as a function of the fill level (s, vertical). In a similar manner, the figures for the functions of s to commodity demand, and s to internal cost can be constructed.

Over time, the system can traverse through multiple pieces of the piecewise constant function, which can be inefficient to evaluate. Due to the structure of these functions, for a given time interval length τ, we can derive a system response model which is piecewise linear in the fill level at the begin of the time interval. The algorithm to derive the model maintains a "forward" and a "backward" projection from the left-hand ($t = 0$) and the right-hand ($t = \tau$) axis, respectively, and each iteration increases at least one of the projection points to the following segment. The algorithm to determine this function runs in $\mathcal{O}(n_{s'})$, where $n_{s'}$ is the number of pieces in the product automaton response function. Whereas the algorithm is trivial in time complexity, the handling of all combinations of fill rate transitions (positive to negative, negative to zero, etc.) is cumbersome. We omit the algorithm for brevity.

Figure 6 illustrates with an (exaggerated) example of the *Charge* mode of a battery how the preprocessing represents the piecewise constant fill level dynamics as a piecewise linear function in the fill level, given a static time interval length τ. The arrows show the fill level projections at the break points of the piecewise functions.

Although the system response also depends on the given demand for the buffer, we choose not to include this demand in the model, because it results in non-(piecewise)-linear equations. Instead, we approximate the buffer demand effect in the DP. The approximation may be improved by computing the response for a set of reference buffer demand levels, and choosing one of these references for each DP time interval.

Next to the continuous system response, we can also precompute the discrete system response, which we use in the DP. For every mode, we discretize the fill levels. Figure 6b illustrates this with ticks on the left hand axis. Each discrete state represents the response in a part of the state space, whereby states do not have to align exactly to these representative states. Some states are infeasible; these are represented by `null`. Furthermore, for each state we need to be able to approximate the DP successor state; since we want to be able to account for demand, we use real (fixed-point) numbers to represent state differences.

b) Expansion: The DP state space expansion starts from the back at $t = n_t$, and iteratively gives an approximation of the cost function at $t-1$ until we have an approximation of the cost function at $t = 0$, in accordance to the Bellman equation. We use a continuous interpretation of the fill level space, and use linear interpolation to determine the cost between the explicitly considered points. The end state cost function penalizes quadratic distance from the start fill level to limit the horizon effect.

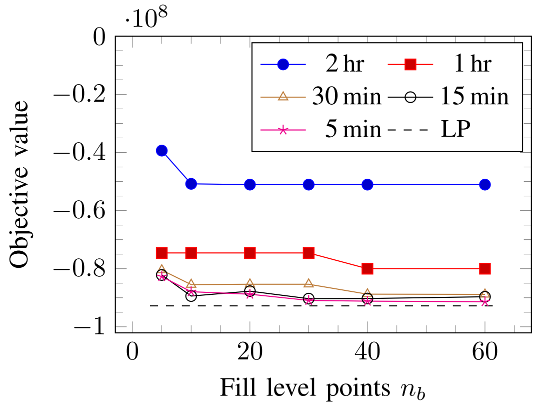


Figure 7. Objective value (lower is better) as a function of n_b and τ.

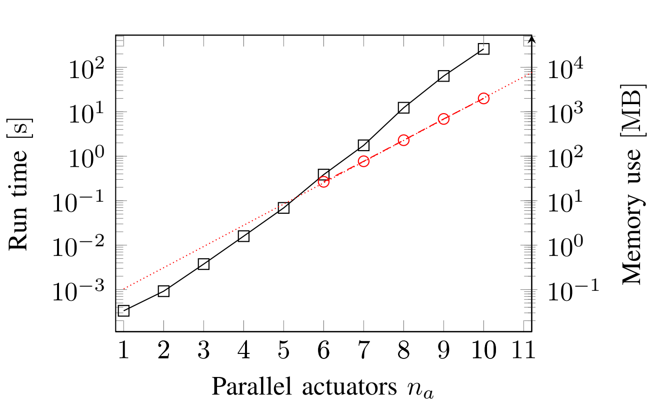


Figure 8. Computational load (left axis, black) and memory use (right axis, dotted red) as a function of n_a combined actuators ($n_t = 96$, $n_b = 30$). Note that the scale on both vertical axes is logarithmic.

The expansion of a time interval consists of two sequential phases: a transition phase and a passage-of-time (τ) phase. These phases are evaluated in reverse order. The transition phase evaluates all transition options at a specific point in time, using an iterative approach to determine the optimal sequence of transitions for each mode, from each point. The τ phase applies the effect of staying in a mode from a point, and uses the cost function at the next time interval.

After the approximation of the cost function over time, we simulate the execution of the policy to determine the schedule of mode changes and the commodity demand pattern. TRIANA returns this demand pattern to the client interface. If this pattern is selected for execution, we give the mode change plan to FPAI as an allocation.

4) Unconstrained: The UNCONSTRAINED control space represents a subset of the BUFFER control space class. An UNCONSTRAINED class is different in that it does not have a connected buffer. Also, the use of the class is different, which may in some cases justify a different control approach. We choose not to implement a specialized optimization strategy for the UNCONSTRAINED control space class. Instead, we map UNCONSTRAINED control spaces to BUFFER control spaces. We reduce the fill level space to a single point.

IV. Experiments

The performance of the approach presented in the previous section is sensitive, both in terms of planning performance (the quality of the outcome) and the computational cost (execution time and memory use). We use simulations to quantitatively evaluate (our implementation of) TRIANA on FPAI.

In this section, we focus on the implementation of the device DPs. In particular, we choose to look at the performance of the implementation of BUFFER control space planning, because it dominates all other relevant cases in terms of execution time. The UNCONSTRAINED control space behaves similar in extreme cases, but avoids the fill level dimension. The execution time of the other DPs is negligible, and thereby not of immediate interest. We omit measurements on the reimplementations of the TRIANA core functionalities, since these perform similar to the original implementations in terms of planning quality and do not introduce new computational challenges.

A. Planning Result

1) Experiment: The BUFFER CSC DP has five parameters: the control space, the initial state, the pricing information, the time discretization step τ, and the number of points n_b which are used to discretize the fill level. To experiment with the behavior of the DP implementation, we set up a simple experiment. We use the control space of a battery with variable efficiency, which is initially half-full and idle. We apply a sinusoidal linear price signal with three periods on a day-ahead horizon ($p(t) = 2 + \sin(-3t\frac{2\pi}{24\cdot 60\cdot 60})$, t in seconds, $0 \le t < 24\cdot 60\cdot 60$).

In the experiment, we take τ and n_b as variables. Note that the appropriate values for τ and n_b may change depending on the structure of the control space and the pricing information. For example, τ needs to account for both the dynamics in the pricing and the model. The number of different fill levels n_b should be high enough to accurately represent the cost function and resulting policy.

To give an indication of the quality of the planning, we use a linear programming (LP) model to give a lower bound on the optimal objective value. The LP model does not account for the variable efficiency, and instead assumes the highest efficiency for the full range.

2) Results: Figure 7 presents the results of the simulation experiments. The solution quality is expressed by the objective value of the optimization problem. These measurements confirm that the time interval length and the number of different fill levels should be chosen high enough to represent the dynamics of the model. At some point, decreasing τ and increasing n_b does no longer meaningfully change the decisions. With a decrease in time interval length, the objective value appears to asymptotically approach a value, which is presumably the (true) optimal solution. This value is close to the LP optimal value bound (1.5 % higher).

An increase of n_b does not always lead to better results, especially for smaller values of n_b. The choice of the exact location of the points affects the approximation, in particular near the edges of the state space, which can give worse results. The choice of τ can affect the approximation similarly.

As a trade off between computation time and resource use, we typically choose $\tau = 900\,\text{s}$ (15 minutes) and $n_b = 30$.

B. Planning Resource Use

1) Experiment: To measure the computational resource use of the DP, we provide the algorithm with increasingly complex control space instances. We generate these instances by connecting copies of a 3–state actuator to the same BUFFER. We denote the number of actuators by n_a.

To measure the memory use, we take the memory use as reported by the Java runtime after a forced garbage collection. Because the time overhead of this forced collection is substantial (i.e. milliseconds), we start the memory measurements from $n_a = 6$. We also provide a lower bound on memory use by analyzing the data structures. Each automaton state gives $n_t \cdot n_b$ DP states, each of which takes up 12 bytes in arrays on a 64 bit system (a reduction to 4 bytes is possible). For the experiment, this means that each automaton state costs at least 35 kB. The number of states in the product automaton is 3^{n_a}.

The experiments are performed on a 4–year old laptop PC running Linux with an Intel Core i5 M540 (2 cores, 4 threads) and 4 GB of RAM. The planning uses a single CPU thread.

2) Results: Figure 8 (prev. page) presents the measurements of the run time and memory use for different numbers of parallel actuators. The exponential growth pattern is evident. There is a small fixed run time overhead in the planning and logging, which is visible at $n_a = 1$, but after that the DP expansion dominates. The run time grows with approximately a factor 5 per actuator. In contrast, memory use grows with a factor 3 per actuator. The measured memory use values correspond well to the estimates given above.

There are no serious jumps in run time, which suggests that the growth in memory use does not result in significant thrashing. An explanation for this is that the DP uses relatively small segments of memory at the same time, which makes caching effective. There is a slight jump from $n = 7$ to $n = 8$ (factor 7), which we believe to come from excessive garbage collection. We expect that disk thrashing occurs if the problem is so large that a single DP phase (time interval) does not fit in the RAM. The PC runs out of memory at $n_a = 11$.

3) Evaluation: For the control spaces we anticipate for practical use, the presented DP formulation has an acceptable cost in run time and memory use. The extensive preprocessing helps to make the DP expansion efficient. The planning can be considered to operate in reasonable time for interactive control from the perspective of TRIANA up to about $n_a = 6$. The device control space planning should be fast, because TRIANA's group planning uses the device planning several times, such that it can choose from multiple possible plannings for the device. For larger instances, the planning is still useful to find a better "baseload" pattern for the devices with a more manageable structure, until memory is exhausted.

The experiments show that in the used DP formulation, estimating the memory use is easy, and run time can be estimated as well. Depending on the circumstances, a controller can make tradeoffs in the parameters of the planning, i.e. it can choose between a fast and an accurate planning. A "smart" controller which manages the use of the planning algorithms is left as future work.

While the experiment only considers concurrent automata as the source of exponential growth in computation time and memory use, timers give an exponential increase in both as well (with a factor 2). Therefore, control spaces with many timers are intractable for this DP formulation. To support such control spaces, a future version may use more coarse abstractions, for example by combining similar timers, at the cost of accuracy.

V. Conclusion

FPAI is a flexible platform for the implementation of demand side management software. Conversely, the TRIANA DSM concept adapts well to an environment that targets a practical setting. We demonstrate this with a prototype of an implementation of TRIANA on FPAI.

Although the control space approach results in complex scheduling problems and introduces approximation inaccuracies, we believe that this impact is limited and that the reduction in implementation and maintenance effort justifies this penalty.

Future work for TRIANA and FPAI includes validation in larger and real world scenarios, various implementation improvements, and market integration.

Acknowledgment

The authors thank Stergios Dagioglou for his experiments with TRIANA for an earlier version of FPAI [14], Bert Claessens (VITO/EnergyVille) for bringing [12] to our attention, and Philipp Wolfrum (Siemens) for discussions on buffer scheduling in relation to [8].

References

[1] OpenADR Alliance, "OpenADR 2.0 profile specification, B profile," January 2013, document 20120912-1.

[2] FlexiblePower Alliance Network, "Being smart with energy." [Online]. Available: http://flexiblepower.org/ [Accessed: 2015-03-12]

[3] A. Molderink, V. Bakker, M. G. C. Bosman, J. L. Hurink, and G. J. M. Smit, "Management and control of domestic smart grid technology," *IEEE Transactions on Smart Grid*, vol. 1, no. 2, pp. 109–119, Sept. 2010.

[4] ZigBee Alliance, "Home automation public application profile 1.2," June 2013, document 05-3520-29.

[5] openHAB, "Empowering the smart home." [Online]. Available: http://www.openhab.org/ [Accessed: 2015-03-12]

[6] ZigBee Alliance and HomePlug Alliance, "Smart energy profile 2 application protocol standard," April 2013, document 13-0200-00.

[7] Z-Wave Alliance, "Home energy management." [Online]. Available: http://z-wavealliance.org/energy-management/ [Accessed: 2015-03-12]

[8] P. Wolfrum, M. Kautz, and J. Schäfer, "Optimal control of combined heat and power units under varying thermal loads," *Control Engineering Practice*, vol. 30, pp. 105–111, September 2014.

[9] K. Kok, "The PowerMatcher: Smart coordination for the smart electricity grid," Ph.D. dissertation, Vrije Universiteit Amsterdam, July 2013.

[10] OpenADR Alliance, "OpenADR 2.0 profile specification, A profile," August 2012, document 20110712-1.

[11] FlexiblePower Alliance Network, "GitHub source repository." [Online]. Available: https://github.com/flexiblepower/ [Accessed: 2015-03-12]

[12] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*, Automation and Control Engineering. Taylor & Francis Group, 2010.

[13] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[14] S. Dagioglou, "An implementation of the planning phase of TRIANA using the FlexiblePower Application Infrastructure," Master's thesis, University of Twente, August 2014.