Available online at www.sciencedirect.com

ScienceDirect

Technology Procedia 00 (2011) 000–000

Technology Procedia

www.elsevier.com/locate/procedia

2nd International Conference on System-integrated Intelligence SysInt2014: New Challenges for Product and Production Engineering

Towards Synthesizing Energy-Efficient Controllers for Modern Production Systems from Scenario-Based Specifications

Joel Greenyer[a], Christian Hansen[b], Jens Kotlarski[b], Tobias Ortmaier[b], a*

[a]*Software Engineering Group, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany*
[b]*Institute of Mechatronic Systems, Leibniz Universität Hannover, Appelstraße 11a, 30167 Hannover, Germany*

Abstract

Due to continuously rising electricity prices, the energy efficiency of production systems is an increasingly important factor in industrial manufacturing. One goal is therefore to reuse the braking energy of manipulator axes. The acceleration and deceleration phases of axes must be synchronized such that, ideally, the braking energy can be reused immediately within the system. In our current technique for the energy-efficient motion planning of concurrent movements of axes, we do not yet consider changing the discrete control logic to reorder movement sequences. This is difficult, since this reordering must not violate critical system requirements. In this paper, we outline a new technique for automatically synthesizing energy-efficient discrete controllers from a scenario-based specification of a production system, which we enrich with information about consumed and generated energy. Last, we provide an outlook on challenging open research problems.

*Keywords: e*nergy-efficiency, multi-axis robot systems, production systems, specification technique, scenario-based discrete behavior specification, controller synthesis, trajectory generation

* Corresponding author. Tel.: +49 511 762 3361.
E-mail address: greenyer@inf.uni-hannover.de.

1. Introduction

Due to continuously rising electricity prices, the energy efficiency of production systems is an increasingly important factor in industrial manufacturing. One goal is therefore to reuse the braking energy of manipulator axes in production machines, for example to accelerate other axes in the system. Today's servo drives already generate electrical energy during deceleration, but this energy is usually dissipated via brake resistors. Instead, we can use the existing DC link coupling of different axes to exchange the electrical energy among them. The challenge is, however, to coordinate the acceleration and deceleration phases of the manipulator axes in such a way that, ideally, the braking energy can be reused immediately and completely within the system.

We have developed different techniques for the energy-efficient motion planning of multi-axis drive systems [10, 11, 12]. Given a set of contemporary point-to-point movements of multiple axes of one multiple robots, we can align the motion trajectories of these axes in an energy-optimal way. To do this, we model the manipulators dynamics and energy consumption- and generation profiles of axis movements, including also the friction, further energy losses and the effect of electrical energy exchange. We then combine the respective axis profiles into a cost function, for which we can automatically find a minimal solution. We currently assume that the contemporary axis movements for which we perform this procedure can be derived from a given discrete control logic of the overall system.

However, to fully exploit the potential of energy reuse, we have to look beyond only sets of single point-to-point movements and also consider the reordering of movement sequences. But this is a difficult task, since it must be ensured that this reordering does not violate critical system requirements. In fact, reordering movement sequences may amount to a complete redesign of the discrete control logic. This is usually avoided in practice, since it implies a cost- and time-intensive iteration in the system's design, re-implementing its PLC code, and repeated testing.

We therefore currently investigate a new methodology for designing energy-efficient production systems. Our vision is to combine the detailed motion planning with an approach for automatically synthesizing energy-efficient discrete control logic from a scenario-based specification of the system.

We propose that engineers specify the discrete behavior of the production system using Modal Sequence Diagrams (MSDs) [6], which allow engineers to intuitively specify what the system should, must, or must not do in certain situations. MSDs have a precise semantics, and we can even simulate the behavior emerging from the interplay of the different scenario descriptions during the system design phase [7]. Furthermore, it is possible to automatically synthesize controllers from these specification; we developed efficient algorithms for this purpose [4]. These algorithms, in particular, not only support specifications of scenarios that describe how the system must react to events in the environment, but support also scenarios that describe assumptions on how the environment may or will behave. This is especially important in mechatronic systems where software controls physical/mechanical processes that adhere to certain laws and principles. MSDs, for this purpose, have also been integrated in the CONSENS specification technique [1, 3].

Once the overall behavior of the production system is specified using MSDs, this specification can be enriched with abstract information of the energy consumed and generated by movement actions. An extended controller synthesis algorithm can then find an energy-efficient control strategy.

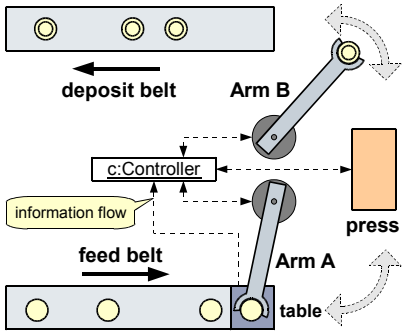
The key advantage of MSDs is not only that they provide intuitive, but precise means for specifying the behavior of a system, but also that they do not force the engineers to over-specify the behavior of the system and, in our case, unnecessarily fix one particular sequence of axis movements when others are also possible. This allows an algorithm to search for energy-optimal control strategies.

Given a synthesized discrete controller, we can then apply the above-mentioned detailed motion planning. The results can finally be combined to an energy-efficient controller of the system and its axes.

In this paper, we describe our proposed design methodology, outline an energy-aware extension of our controller synthesis procedure, and identify open, interdisciplinary research challenges. This paper is structured as follows. In Sect. 2, we explain the foundations and introduce a running example. Then we explain our design methodology and outline the extension of our controller synthesis algorithm in Sect. 3. We report on related work in Sect. 4 and, last, we summarize and give an outlook in Sect. 5.

2. Foundations

As a running example, we consider a production system consisting of two arms and a press (see Fig. 1). One arm, arm A, picks up metal blanks that arrive from a feed belt on a table and places them into a press, where they are pressed into plates. The other arm, arm B, picks up the pressed plates and places them on a deposit belt, where they are transported off again. The behavior controlled by a software controller that receives events from a sensor in the table and controls the arms and the press. The arms and the press are actor/sensor components; for example, the controller can order arm A to move to the press, and arm A will notify the controller when it arrives.



Arm A:

R1) When a blank plate arrives at the table, Arm A must pick it up, move it to the press, and release it into the press. The arm then has to move back to the table, where it must arrive before the next blank arrives.

A1) If Arm A is ordered to move to the press, it will eventually arrive at the press

A2) If Arm A is ordered to move to the table, it will eventually arrive at the table

A3) If Arm A picks up a blank, moves to the press, and returns to the table, the next blank will not arrive before Arm A has returned to the table.

Press:

R2) When Arm A releases the blank into the press, the press must press. The next blank must not be released before the pressing finished.

A4) The press finishes pressing before Arm A returns to the table.

Arm B:

R4) After the press has finished pressing, Arm B must pick up the pressed plate and move to the deposit belt... [*further specification omitted for brevity*]

Fig. 1. Sketch of the production cell and excerpt from a textual specification [4]

Fig. 1 shows a sketch of the production cell along with parts of a textual specification of its software controller. The software controller can control when it sends which commands to the actors, but it cannot control events from the sensors. For example, the controller can order arm A to move to the press, but it cannot control when arm A will arrive there or if it will arrive there before the next blank arrives. In fact, the controller cannot even control whether arm A will arrive at the press at all. From the perspective of the software controller, the table sensor, press, and arms are the *environment*, and the events coming from this environment are *uncontrollable*. However, the behavior of this environment cannot be arbitrary if we want the software controller to fulfill its requirements. We have to make *assumptions* on how the environment may, will or will not behave. One assumption is that, if arm A is ordered to move to the press, it will eventually arrive there (A1). This assumption has to be guaranteed by the supplier of the robot arm component or acts as a requirement for the engineers constructing this component.

2.1. MSD Specifications

MSDs are a formal interpretation of sequence diagrams proposed by Harel and Maoz [6], based on the concepts of Live Sequence Charts (LSCs) [2]. An MSD specification describes the valid interaction behavior of components, which we more generally call *objects* in the following. The set of interacting

objects is called the *object system*. The objects interact by sending messages among each other. A message has a name and one sending and one receiving object. In this paper, we assume that the sending and the receiving of a message together to form a single *message event* or simply *event*, i.e., the messages are *synchronous*. The *behavior* of a system is its set of possible sequences of events. A single sequence of events is also called the *execution* or *run* of the system. As we consider systems that can, in principle, run indefinitely long, we consider runs to be infinite.

Objects can either be *system objects* or *environment objects*. Messages events sent from system objects are *controllable*, messages events sent from environment objects are *uncontrollable*. In our example, we consider the software controller to be a system object; the table sensor, the press and the two arms are environment objects. We model the object system using UML composite structure diagrams (bottom left of Fig. 2), which are related to internal block diagrams of SysML or active structure diagrams of CONSENS [1]. The objects are instances of classes in a class diagram (top left of Fig. 2).

An MSD has *lifelines* that each represents exactly one object in the object system. A message in an MSD, also called a *diagram message*, has a name and a sending and a receiving lifeline. The messages in an MSD have a *temperature* and an *execution* kind. The temperature can be either *hot* or *cold*; the execution kind can be either *monitored* or *executed*. In our figures, we label the messages accordingly with (c/m), (c/e), (h/m), and (h/e). In addition, hot messages are colored red; cold messages are colored blue. Executed messages have a solid arrow; monitored messages have a dashed arrow.

On the right of Fig. 2, we see two MSDs that model requirement R1 and assumption A1. Intuitively, the semantics of an MSD is follows. An MSD is activated if a message event occurs in the system that matches the first message in the diagram; we also say, an *active copy* or *active MSD* is created. Then, this active MSD progresses as message events occur in the system as described in the MSD. If the progress reaches a message that is monitored, this message may or may not occur. If the message is executed, the message must eventually occur. We call this a *liveness* requirement. If the message is hot, messages are forbidden to occur that the scenario specifies to occur only earlier or later. We call this a *safety* requirement. If the message is cold and a message occurs that is specified to occur earlier or later, this "aborts" the progress of the MSD. Messages that are not specified in the MSD are ignored, i.e., they do not influence the progress of the MSD and the MSD does not impose requirements on them. The active MSD also terminates when it progressed until the end of the diagram.

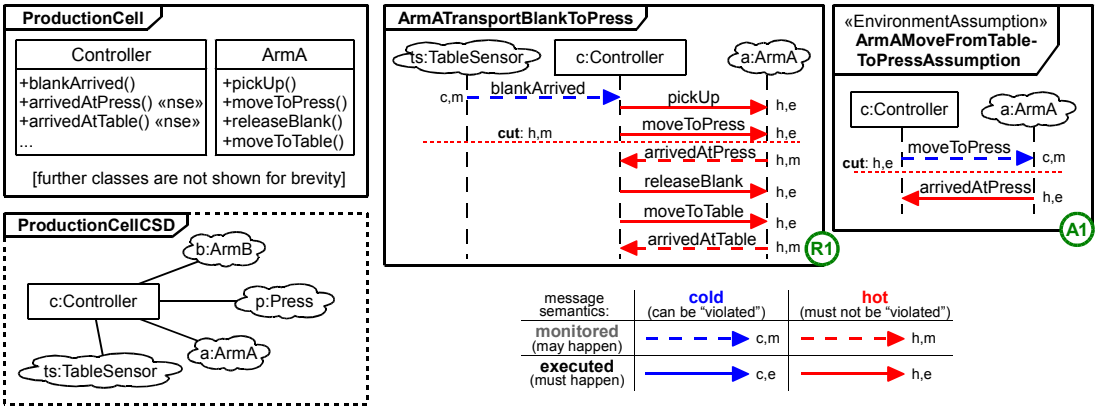


Fig. 2. Class- and Composite Structure Diagram and MSDs from the production cell specification [4]

The above-mentioned progress of the MSD is represented by the *cut*, which spans all lifelines and marks messages that occurred. Messages immediately in front of the cur are called *enabled*. In Fig. 2, cuts are shown as a dashed horizontal line. In the MSD *ArmATransportBlankToPress* the message *arrived-AtPress* is currently enabled. Note also that multiple MSDs can be active at the same time and, if they have the same messages enabled, they progress synchronously on the occurrence of this event.

Let us look at the MSD *ArmATransportBlankToPress* in a bit more detail. This MSD is activated when the table sensor notifies the controller about the arrival of a blank. The controller must then order arm A to pick up the blank and move to the press. (We assume that arm A is at the table in the initially.) Then the controller must wait until the arm arrives at the press (this state is shown by the cut in Fig. 2). Since this message is monitored, it would be fine for the arm never to arrive at the press, but since it is hot, no other event in the diagram must occur; for example, no blank must occur while the MSD is in this cut. After the arm arrives at the press, it must release the blank, move back to the table, and then the controller must again wait until arm A arrives at the table. Throughout this process, no blank must arrive.

In order for the controller to be able to realize this requirement, we must assume that arm A, if told to move to the press or back to the table, will also do so. Also, we must assume that indeed no new blanks will arrive while the arm is performing the described operation. These assumptions are formulated in A1, A2, and A3 (see Fig. 1). Fig. 2 shows the MSD *ArmAMoveFromTableToPressAssumption*, which models assumption A1. MSDs that model assumption are called *assumption MSDs* and have an additional label *«EnvironmentAssumption»*; MSDs that model requirements are called *requirement MSDs*.

If there are missing assumptions or contradicting liveness and safety requirements, it may very well be that a specification is *unrealizable*. We call an MSD specification *realizable* if there exists an *implementation* of the system objects that for all possible executions with all possible environment objects satisfies all requirement MSDs, assuming that all assumption MSDs are satisfied. An execution satisfies an MSD if it leads to no violation of any of the liveness or safety properties that it describes. For a more precise explanation of the MSD semantics, we refer to previous work [6, 3].

2.2. Simulation and Controller Synthesis

During the design, it is crucial to check the realizability of an MSD specification. A first way to analyze an MSD specification is by executing it via the *play-out algorithm* [7, 20]. The play-out algorithm allows engineers to simulate the behavior emerging from the interplay of the specified scenarios. Roughly, play-out works as follows. As environment events occur, this activates MSDs or progresses previously activated MSDs. If this leads to a state where in one or more active MSDs executed system messages enabled, one of these is selected non-deterministically for execution unless it is forbidden in another active MSD. Executing a message event will again progress active MSDs or activate new active ones. We keep executing system messages as long as there are no more enabled executed system messages. Then we wait for the next environment event to occur, and this process is repeated.

We assume that the controllable system objects are always faster than the environment. Therefore, we assume that the controllable system objects can always perform any finite number of steps before the next environment event occurs. Of course, in a time-critical system, this assumption must be revised. There are also timed extensions of MSDs and LSCs [8, 5].

When executing an MSD specification with play-out, it can happen that a deadlock state is reached where executed system messages are enabled in some active MSDs, but they are all forbidden by others. This indicates that the specification may be unrealizable, but it could also be that, by making some other non-deterministic choice some steps before, this situation could have been avoided.

There also exist algorithms for finding out automatically if an implementation or a valid play-out behavior of an MSD specification exists. Different approaches have been proposed in the past that can

automatically find finite-state machines that resemble a valid implementation of an MSD specification [18, 19]. Synthesizing a valid controller can require to explore all possible configurations of active MSDs and their cuts. This state space grows exponentially with the number of MSDs in the specification. We have lately developed an efficient on-the-fly controller synthesis algorithm that often only explores parts of this state space [4]. This algorithm considers the controllable system objects and the uncontrollable environment objects to be two players that play an infinite game. The possible moves of the players are sending system resp. environment messages. The system "wins" if it can always fulfill all safety and liveness properties of all requirement MSDs provided that the environment never violates the safety or liveness properties of any assumption MSD.

Fig. 3 illustrates part of the so-called *specification state graph* that is explored during controller synthesis. Each state is labeled with the active MSDs and their cuts in that state. The numbers in braces behind the MSD names describe the cut position on each lifeline. Transitions between states are labeled with message events. Transitions with uncontrollable (environment) events are shown as dashed arrows. Initially (state 1), no MSDs are active. The event *blankArrived* activates the requirement MSD *ArmA-TransportBlankToPress* (see Fig. 2) and the assumption MSD *NoBlankArrivesBeforeArmAReturned-ToTable* (modeling A3, see Fig. 1). The events *blankArrived* and *arrivedAtTable* in state 4 lead to a safety violation in the requirements, but also in the assumptions.

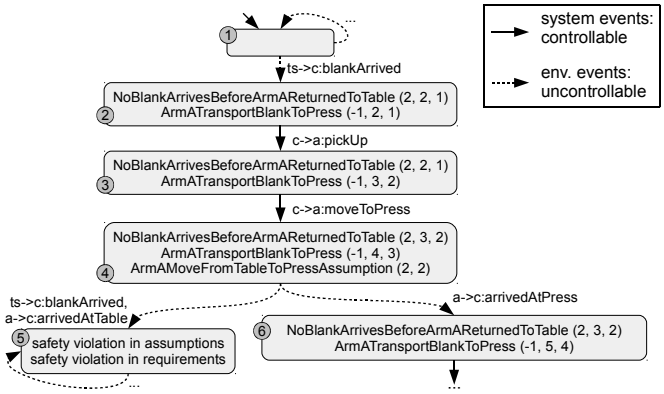


Fig. 3. Part of the specification state graph explored when synthesizing the controller for the production cell MSD specification.

2.3. Modeling and Analysis of Energy Flow in Multi-Axis Drive Systems

We have elaborated different methods of movement optimization for energy efficient manipulator systems [10, 11, 12]. In this example, the energy demand improvement potential for the given production cell system is demonstrated by energy demand simulations of two different control scenarios. First, we assume that the two manipulator arms A and B of the example application (compare section 2.1) are actuated using conventional servo drives as shown in Fig. 4, where each arms is attached to a servo motor (M_A, M_B; e.g. permanent magnet synchronous motors), optionally via transmission gears (ratios u_A and u_B). The actuator motions are controlled by servo inverters (I_A and I_B) which are connected to one common DC link. The shared supply module (S) rectifies a three-phase grid voltage to generate DC voltage and, hence, supplies both servo inverters (P_{sup}). Since servo drives are capable of energy recuperation from the mechanical movement, e.g. during deceleration of an arm, electrical energy is exchanged between the applied axes. Energy surpluses are dissipated via the brake resistor (P_{chp}) since the DC link internal capacitances and the component's voltage strength are limited.

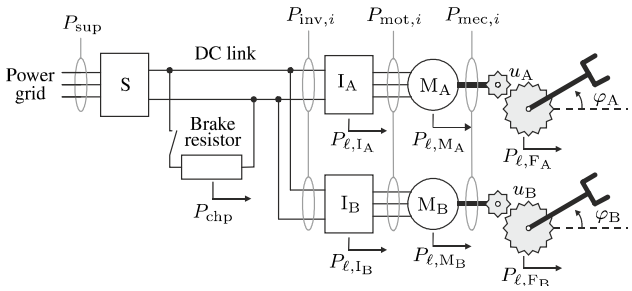


Fig. 4. (a) Electrically coupled servo drive system with two manipulator arms

For the execution of the pick and place tasks to move the blanks plates from the feed belt to the press using arm A, and, after pressing, to the deposit belt using arm B, point-to-point (PTP) trajectories with trapezoidal velocity profiles [9] are applied. The resulting movements each exhibit three separate phases: acceleration, constant velocity, and deceleration phase. With consideration of friction losses in the mechanics and further energy losses of the motors and inverters, the electrical power demand of each inverter can be calculated [11]. Finally, the energy demand of the supply module depends on the coordination of both arm movements as shown in the following.

The energy saving effect of energy exchange amplification is illustrated by simulation result presented in Fig. 5. Here, two different scenarios of coordinated PTP movements of both arms are given. Scenario 1 presents an inappropriate sequence of arm movements, since no direct exchange of electrical energy appears, while in scenario 2, the deceleration phase of arm A (power recuperation) and the acceleration phase of arm B (power demand) have maximum time overlap, resulting in a maximized exchange of electrical energy via the DC link. Hence, the energy demand for acceleration of arm B is supported by the DC link internal energy exchange, resulting in reduced energy demand from the supply module. Additionally, since the recuperated energy by arm A is used for arm B's acceleration demand, less energy surpluses must be dissipated by the brake chopper. Consequently, in scenario 2, the total supply energy demand of the multi-axis complete system is reduced by about 20% for the chosen system parameters, compared to the uncoordinated movement in scenario 1. The given examples highlight the importance of suitable manipulator movement coordination within the considered production cell.

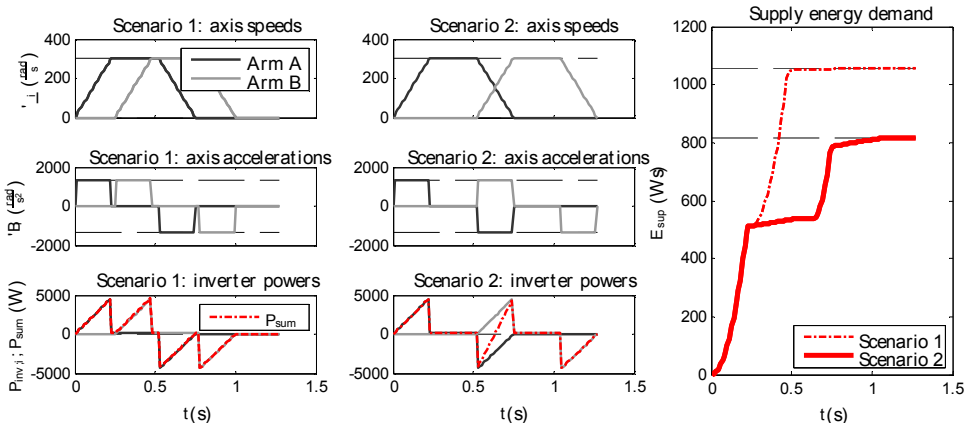


Fig. 5. Manipulator arm movement profiles: motor axis angular speed $\dot{\varphi}_i$, acceleration $\ddot{\varphi}_i$, electrical inverter power $P_{\mathrm{inv},i}$, summarized inverter powers P_{sum} and the trends of supply energy E_{sup} for both scenarios 1 and 2

3. Design Methodology and Energy-Aware Controller Synthesis

The two techniques mentioned in the previous section operate on two different levels of detail and are applied in different development phases. The MSD specification is modeled early for a first specification of the relative order of discrete events in the system. From this specification, if sufficiently detailed, a discrete controller for the overall system can be derived. The energy-efficient motion planning is performed later, when more detailed information about, e.g., the robot arms, friction, masses, etc. is available. It considers a detailed model of the axes' movements in individual time windows that each correspond to certain sub-sequences in the above-mentioned discrete control logic of the system.

The problem is that the detailed motion planning does not consider potentially more efficient orderings of axis movements outside the considered time windows. The MSD specification of the discrete overall system behavior, on the other hand, does not consider energy consumed or generated during acceleration and deceleration of manipulator axes.

Our goal is to bridge this gap between the two procedures. We want to allow engineers to consider energy consumption already during the conceptual design phase and thereby enable them to more drastically optimize the energy-efficiency of production systems and other mechatronic systems.

In order to achieve this goal, our first approach is to refine our MSD specification with approximate information about the energy consumed and generated by the individual axis movements. At an early design stage, detailed information may not be available, but viable assumptions can often be made.

MSDs allow us to comfortably refine our specification by simply adding MSDs that can extend and restrict the previously specified behavior. We add MSDs that require that the command for an axis to perform a movement is followed by detailed commands for the axis to start and stop accelerating, and then start and stop decelerating, whereby certain amounts of energy are consumed and generated. For our example, we introduced the MSDs shown on the left of Fig. 6. The MSD *ArmAMoveToPressAccAndDec* specifies that after sending *moveToPress* to arm A, the controller must also send the messages *beginAcc*, *endAcc*, *beginDec*, *endDec* to arm A. The messages *beginAcc* and *beginDec* are parameterized to attach information about the energy consumed and generated during acceleration and deceleration, respectively.

We specify integer values that resemble a rough estimate on the expected amounts of energy consumed and generated. Here we assume that deceleration of arm A releases half the energy (5) as consumed for acceleration (10) due to energy losses, such as friction. For the movement of the arm without payload, e.g., arm A moving back to the table, we specify different parameters in an extra MSD.

This information can now be used by the controller synthesis algorithm. To do this, we first calculate how much energy will be consumed when we arrive in a certain state.

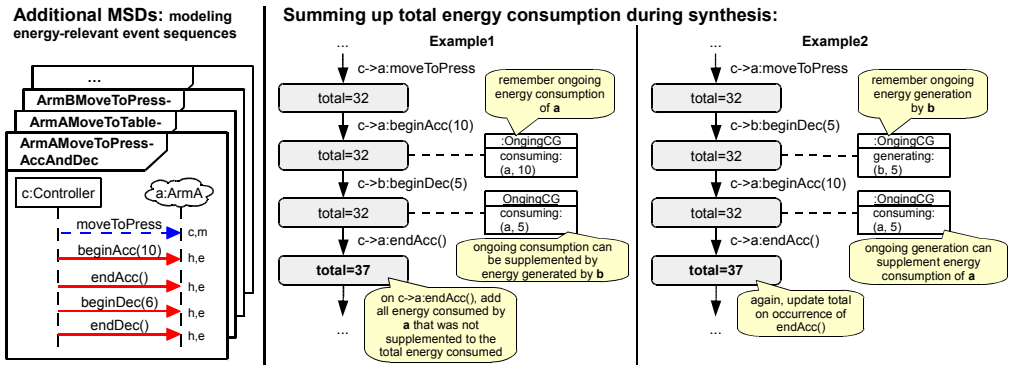


Fig. 6. Additional MSDs specify energy consumption/generation; total energy consumed up to a particular state can be calculated.

We calculate these values for each state as shown in the middle and right of Fig. 6. Let's consider the calculation shown in the middle: if a *beginAcc* message was sent, we remember the energy that this acceleration consumes in a data structure *OngoingCG* ("ongoing consumptions and generations"). This structure is attached to every state in which there are ongoing consumptions/generations; it comprises of two maps (Object → Int) consuming and generating, which store the currently consumed and generated amounts of energy per axis. If a *beginDec* message occurs (for another axis, here arm B) before the former acceleration phase (*endAcc*), the energy generated by the deceleration can be deduced from the energy consumed during acceleration by arm A. This corresponds to the idea that we can apply our detailed motion planning on these intervals where consumptions and generations occur contemporarily. Only after an *endAcc* message occurs, the energy that could until then not be supplemented by a concurrent deceleration is finally added to a *total* of energy consumed from an external source. If the deceleration of an axis begins before the acceleration of another, this works similarly (see right of Fig. 6).

This forward propagation of the aggregated total energy cost can be done while the controller synthesis algorithm explores the specification state graph as shown in Fig. 3. Upon finding possible control strategies, the question is now how to select the most energy-efficient strategy.

We currently assume that, although the execution of the system is in principle infinite and cyclic, that we have non-cyclic paths between events that represet the initialization and termination of reward-generating processes in the system. In the case of our example system, this is the arrival of a blank (initializing) and Arm B releasing the processed item into the deposit belt (terminating). There can be many different paths between these two events, since environment events may occur in different orders and the system may have different choices of how to react. Along these paths, starting from terminal states, we can back-propagate the information about the energy costs: In states where the system can make a choice, we remember the best choice and back-propagate the corresponding energy cost; where different events can occur in the environment, we assume the worst and back propagate the most expensive cost. This follows the minmax decision rule, which is a classical game-theoretic principle.

We are investiating how to extend this approach if we do have cycles within reward-generating processes and how to include time and optimize the tradeoff between time- and energy-efficiency. Also, instead of always assuming the worst that can happen in the environment, we would like to consider what is *most likely* to happen in the environment. When we assign probabilites to environment events, however, we are dealing with stochastic games. We currently investigate the applicability of existing algorithms for this problem.

4. Related Work

The energy-efficient motion planning is also considered e.g. by Pellicciari et al. [13] who apply time scaling of individual robot trajectories by utilization of idle times of the manipulator if such exist after the considered tasks. Wigström et al. present a dynamic programming method for task scheduling in a multi robot cell [14]. Here, optimal time scaling factors for each task are determined to generate individual energy optimal trajectories. However, the approaches only consider the isolated time windows of single manipulator tasks, or, as given in the second example, a selective exploitation of the DC link coupling of different manipulation systems is not integrated, e.g., to utilize energy exchange for further demand reduction.

To the best of our knowledge, the synthesis of energy-efficient discrete controllers for production system has not yet been investigated. UPPAAL TIGA implements algorithms for synthesizing time-optimal strategies in real-time systems that can be modeled using timed game automata [17]. These models could also be extended with a notion of consumed energy. However, we believe that our scenario-based modeling technique provides a more intuitive and approach for early behavior specifications.

5. Conclusion and Future Work

Energy-efficiency is an increasingly important aspect of modern production system and reusing the braking energy of manipulator axes can significantly reduce their energy-consumption. There already exist techniques for the detailed energy-efficient motion planning of manipulators, but these techniques only regard isolated time windows within often complex processes of the overall production system. We outlined an approach for considering the energy-efficiency also on the level discrete control logic of the overall system in order to more drastically exploit the energy-saving potential present in many systems. This approach comprises a flexible, but precise scenario-based technique for specifying the system behavior during the early system design and automatic methods for synthesizing energy-efficient control strategies. In these strategies, we maximize contemporary acceleration and deceleration phases of axes for which we can then later apply the detailed motion planning. We are currently elaborating and implementing these techniques based on an existing tool infrastructure.

The future research direction lies in investigating how to include more detailed information in the scenario-based specification and consider this information also in the synthesis procedure; we currently see timing aspects and probabilities as most relevant. Automatic controller synthesis is a computationally complex task, but we are optimistic that it can be applied in practice by using on-the-fly algorithms [4], compositional techniques [5] or heuristics.

One major question is how to align the two described techniques in an incremental development scenario, where results from the detailed motion planning can be used to refine the specification of the discrete control logic and then apply an incremental controller synthesis. Synthesis and motion planning techniques can in principle even be applied at runtime, which would allow systems to self-optimize their behavior when they receive changing tasks or are deployed in changing environments.

References

[1] Anacker H, Brenner C, Dorociak R, Dumitrescu R, Gausemeier J, Iwanek P, Schäfer W, Vaßholz M. Methods for the Domain-Spanning Conceptual Design. *Design Methodology for Intelligent Technical Systems*, Springer; 2014, p. 117–182

[2] Damm W, Harel D. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, Vol. 19, 2001, p. 45–80

[3] Greenyer J. Scenario-based Design of Mechatronic Systems. *PhD Thesis*, University of Paderborn, Paderborn, 2011

[4] Greenyer J, Brenner C, Cordy M, Heymans P, Gressi E. Incrementally Synthesizing Controllers from Scenario-Based Product Line Specifications. *Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013, p. 433–443

[5] Greenyer J, Kindler E. Compositional Synthesis of Controllers from Scenario-Based Assume-Guarantee Specifications. *Proc. of the ACM/IEEE 16th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS 2013)*, 2013

[6] Harel D, Maoz S. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software and Systems Modeling (SoSyM)*, Vol. 7, 2008, p. 237–252

[7] Harel D, Marelly R. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*, Springer, 2003

[8] Harel D, Marelly R. Playing with Time: On the Specification and Execution of Time-Enriched LSCs, *Proc. Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'02)*, 2002, p. 193–202

[9] Biagiotti L, Melchiorri C. *Trajectory Planning for Automatic Machines and Robots*. Springer, 2008, p. 62–76

[10] Hansen C, Öltjen J, Meike D, Ortmaier T. Enhanced Approach for Energy-Efficient Trajectory Generation of Industrial Robots. *Proc. of the 2012 IEEE Int. Conf. on Automation Science and Engineering*, 2012, p. 1–7

[11] Hansen C, Kotlarski J, Ortmaier T. Path Planning Approach for the Amplification of Electrical Energy Exchange in Multi Axis Robotic Systems. *Proc.of the 2013 IEEE Int. Conf. on Mechatronics and Automation*, 2013, p. 44–50

[12] Hansen C, Kotlarski J, Ortmaier T. Experimental Validation of Advanced Minimum Energy Robot Trajectory Optimization. *Proc. of the 16th Int. Conf. on Advanced Robotics*, 2013

[13] Pellicciari M, Berselli G, Leali F, Vergnano A. A method for reducing the energy consumption of pick-and-place industrial robots. *Int. Journal on Mechatronics*, Vol. 23, 2013, p. 326–334

[14] Wigström W, Lennartson B, Vergnano A, Breitholtz C. High-Level Scheduling of Energy Optimal Trajectories. IEEE Transaction on Automation Science and Engineering, Vol. 10, No. 1, 2013, p. 57–64
[15] Ferri C, Viescas A, Moreshet T, Bahar RI, Herlihy M. Energy efficient synchronization techniques for embedded architectures. *Proc. of the 18th ACM Great Lakes symp. on VLSI*, 2008, p. 435–440
[16] Jozwiak L; Gawiowski D, Slusarczyk A. Multi-objective Optimal Controller Synthesis for Heterogeneous Embedded Systems. *Proc. of the Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation*, 2006, p. 177–184
[17] Behrmann, G.; Cougnard, A.; David, A.; Fleury, E.; Larsen, K. G.; Lime, D. Damm, W. & Hermanns, H. (Eds.) UPPAAL-Tiga: Time for Playing Games! *Proc. Int. Conf. on Computer Aided Verification (CAV'07)*, Springer, 2007, 4590, 121-125
[18] Harel, D.; Kugler, H. Synthesizing State-Based Object Systems from LSC Specifications. *International Journal of Foundations of Computer Science*, 2002, 13:1, 5-51
[19] Bontemps, Y.; Heymans, P. From Live Sequence Charts to State Machines and Back: A Guided Tour. *IEEE Transactions on Software Engineering*, IEEE Press, 2005, 31, 999-1014
[20] Brenner, C.; Greenyer, J. & Manna, V. P. L. The ScenarioTools Play-Out of Modal Sequence Diagram Specifications with Environment Assumptions. In *Electronic Communications of the EASST, Proc. Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'13)*, EASST, 2013, 58