HAL
open science

Reactive control of adaptive embedded systems

Eric Rutten

To cite this version:

Eric Rutten. Reactive control of adaptive embedded systems. [Research Report] RR-6604, INRIA. 2008, pp.20. inria-00308660v2

HAL Id: inria-00308660

https://inria.hal.science/inria-00308660v2

Submitted on 1 Aug 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Reactive control of adaptive embedded systems

Eric Rutten

N° 6604

July 2008

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6604--FR+ENG

Reactive control of adaptive embedded systems

Eric Rutten*

Thème COM — Systèmes communicants
Projet POP ART

Rapport de recherche n° 6604 — July 2008 — 20 pages

Abstract:

Embedded systems have to be more and more adaptive: they must perform reconfigurations in reaction to changes in their environment, related to resources or dependability. The management of this dynamical adaptivity, as in autonomic systems, can be considered as a control loop, on continuous or discrete criteria. Embedded systems are also by nature safety-critical, and must be statically checkable for stringent guarantees of predictability. This is the goal of the formal techniques for their specification, validation and verification. The reactive systems approach to this problem exploits models and automated tools based on state machines. This position paper aims at drawing the attention of the autonomic computing community towards the existence of control solutions in discrete controller synthesis (DCS), and of programming languages and analysis and synthesis tools to support them. We use them as a foundation for an approach combining adaptivity and predictability, and describe a method for the safe design of safe execution systems, relying on a technique for the static guarantee of dynamic reconfigurations. We base our position on previous work in reactive and embedded systems, and draw directions detailing opportunities and challenges, towards the model-based control of adaptive systems.

Key-words: Embedded systems, safe design, adaptive systems, autonomic computing, behavior enforcement, self-configuration, discrete controller synthesis, reactive systems, synchronous programming.

* http://pop-art.inrialpes.fr/people/rutten, Eric.Rutten@inria.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Contrôle réactif de systèmes embarqués adaptatifs

Résumé : Les systèmes embarqués doivent être de plus en plus adaptatifs : ils doivent se reconfigurer en réaction à des changements dans leur environnement, concernant les ressources ou la fiabilité. La gestion de cette adaptativité dynamique peut être traitée, par exemple dans le calcul autonomique, au niveau de l'intergiciel, en captant l'état du système, en décidant d'actions de reconfiguration, et en les exécutant. On peut donc considérer qu'il s'agit d'une boucle de contrôle, sur des critères continus ou discrets. Les systèmes embarqués sont aussi par nature à sécurité critique. Obtenir une prévisibilité statique est le but des techniques de spécification, validation et vérification. L'approche des systèmes réactifs exploite des modèles, par exemple synchrones, et des outils automatiques pour la vérification ou la synthèse de contrôleurs discrets (SCD). Dans cet article de position, nous proposons une approche pour combiner adaptativité et prévisibilité, et décrivons une méthode pour la conception sûre de systèmes à exécution sûre, basée sur une technique pour la garantie statique de reconfigurations dynamiques.

Mots-clés : Systèmes embarqués, conception sûre, systèmes adaptatifs, calcul autonomique, synthèse de contrôleurs discrets, programmation synchrone.

1 Context and position statement

Context. Embedded systems are proliferating, in a great variety of environments. Their design involves techniques ranging from application expertise, to software validation, and to System on Chip design.

Embedded systems have to be more and more *adaptive*: they must perform reconfigurations in reaction to changes in their environment concerning e.g., power supply, communication bandwidth, quality of service, or also typically dependability and fault tolerance for a *safe execution*. The run-time management of this **dynamical adaptivity** is the object of research on ways to design and implement adaptation strategies. One approach is in autonomic computing, where functionalities are defined, typically at middleware level, for sensing the state of a system, deciding upon reconfiguration actions, and performing them.

Embedded systems are by nature *safety-critical*, due to their deep interaction with their embedding environment, machines, goods or people, and also due to their wide-spread use and the cost of maintenance or recall of defect products. They have a special requirement for *safe design*, which classicaly contradicted dynamical operating system features. Obtaining **static predictability** is the goal of their specification, validation and verification techniques. The reactive systems approach to this problem exploits finite or bounded models e.g., synchronous, and automated algorithms for verification e.g., by model checking, or synthesis e.g., using discrete controller synthesis (DCS).

Statement. An important concern is then to combine these two different requirements for embedded systems i.e., to be **adaptive and predictable**. This goal can also be formulated as: a method for the safe design of safe execution systems, a technique for the static guarantee of correct dynamic reconfiguration, a tool for the off-line computation of run-time controllers.

In this position paper, we argue that **model-based control of adaptive systems** can contribute to this goal, in the following concrete directions:

1. modeling adaptation domains and reconfiguration mechanisms of autonomic systems in terms of their events and states, as discrete event systems;
2. stating the adaptation strategies and safety features in terms of logical properties;
3. obtaining their correct control using the available techniques for specifying, verifying, testing, compiling, and also, more originally, by solving the DCS problem thus defined;
4. obtaining control solutions adapted to each application, exploiting their specificities, thanks to the automation of the method that offers easy modifiability;
5. integrating these formal techniques in a user-aimed tool and method e.g., through the definition of domain-specific languages, with a compilation involving DCS;
6. and, last but not least, integrating an executable form of the resulting controller into the execution platform e.g., at middleware-level.

The scientific interest of the proposal is manifold. Reconfigurable and autonomic systems could benefit from a set of techniques that are formally-based and mechanical, in the sense that automatic tools are already available for use, even if they can still be improved and adapted. Reactive systems models and methods (e.g. DCS) could benefit from being exposed to new demands, notably concerning: at the front-end, expressivity of models in the direction of timed or value-dependent behaviours; at the kernel, efficiency of computations and algorithms implemented in the tools; and also, at the back-end, execution schemes for the synthesized controllers.

The connection of these areas: formal approaches and middleware-level adaptivity, that have developed independently until then, can benefit to the global corpus of design methods for embedded systems. DCS is an opportunity to introduce formal methods in industrial practice in a user-friendly way, notably through the use of domain-specific languages, familiar to experts of application fields, non-experts of formal methods.

Sustaining previous work. We base our statement not only on general considerations arising from a survey of the literature, on the side of autonomic computing [14, 15] and of discrete event systems and their control [6], but we base it also on experience gathered in previous work, which, although not explicitly related to autonomic computing, appears to have been an approach to similar goals.

The focus was on a more specialized application domain, the area of real-time control systems, particularly the programming of robotic systems. The work aimed at contributing techniques for the correct control of multi-task systems, with guaranteed safety of the embedded system, and easy reuseability and modifiability of applications. It resulted on the one hand in a set of modeling patterns for multi-task systems (typically of resources capacities and constraints, activities and services available in the system, and applications making requests). The model patterns can be used as a library to construct a global model of possible behaviors (as a model of the plant in control theory) and a specification of their correctness (as a control objective in regulation techniques). On the other hand, and on these bases, this work resulted in techniques for the automated generation of correct handlers [19, 20] and property-enforcing layers [1], through the use of discrete controller synthesis techniques, implemented in a symbolic computation tool [18], and connected to the synchronous languages and compilers [1].

A further development of our research is the encapsulation of these results into a domain-specific language framework [7], providing for a tool not requiring the deep formal expertise involved in automata-based models and synthesis algorithms, and wrapping the formal techniques into a compilation-like process. Examples taken from this work are used to illustrate our point in Section 2, and a synthetic account of this work is given below in Section 3.

Organization of the paper. This position paper is organized in order to present material supporting the statement formulated above. It will provide a review of the respective backgrounds: adaptive and reconfigurable systems in Section 2.1, and reactive systems and discrete controller synthesis (DCS) in Section 2.2. It is on these bases that Section 3 for-

mulates our proposal for model-based control of adaptive systems, by precising the position in this paper, detailing different aspects of the question, recalling useful preliminary work, and listing some of the opportunities and challenges involved. Section 4 concludes.

2 Review of the backgrounds

The background of our position lies in each of the two concerned areas of, on the one hand, the middleware-level support of run-time adaptation, reconfiguration and autonomicity, and on the other hand, the formal approach to reactive systems and the techniques of DCS.

2.1 Adaptive, reconfigurable and autonomic systems

Systems have to be adaptive, especially embedded and portable devices, because of the dynamicity of their environment, and in order to avoid manual management of reconfiguration [15]. They have to be equipped with adaptation mechanisms for sensing and changing configuration, and also for deciding upon the reconfiguration, and controlling the adaptation. Autonomic systems and computing [14] are an approach to the design of such automatic adaptation e.g., in the case of self-configuration or self-optimizing.

An example scenario of adaptation can be illustrated by a cellular phone with a video-streaming application, that enables for visioning of video-on-demand programs, or television broadcast. Such an application can perform in several different modes for fulfilling this functionality, differing in:

- levels of quality of service (QoS); in a view related to fault-tolerance, they can also be seen as degradation levels, along a degradation axis [22].

  In the example of display, we can consider:

  - modes: high / medium / low resolution, black and white / colour display, compression level.
  - quality levels: indeed, an interesting feature is then to have "graceful" degradation, with gracefulness amenable to be defined in different fashions, according to criteria from the application domain. Here, criteria have to decide whether black and white in high resolution is better than colour in medium resolution, or whether it is preferable to avoid abrupt quality change but remain average, or to keep quality at the highest.

- resource consumption, that can have quite different characteristics if several algorithms versions of the functionality are available, regarding: computing (in terms of worst-case execution time or CPU load), communication (switching on or off signal compression according to available bandwidth), memory footprint, access time between main and secondary memory, and also energy.

Adaptation policy is, at a different level, also a point of variability. It has to be defined, how to react to changes in the environment, and what choices to make for a new configuration or mode. The general goal is:

Whatever the environment changes, functionality must be fulfilled, at a good level.

In our example, if bandwidth goes down because of signal propagation getting weaker in a badly covered area, then a variety of reactions is possible, including e.g., triggering compression, but then computing time rises, as well as energy consumption. Another reason for reacting can be an incoming message, with an image attachment. If the reception mode involves immediate download, then it could imply degraded video quality in order to free some computing, memory and bandwidth. Or else, reception can be shut off by the user while looking at video uninterrupted and undisturbed, and resumed later. This latter point can illustrate that knowledge of the application state allows for some lookahead in the control.

This example gives an idea of the adaptation criteria, the levels on functionality, the application-specific aspects, the management policies that have to be considered.

Needs for run-time adaptation more generally concern the use of an execution platform by applications, which must be able of variation, as transparently as possible.

- In the case of resource management in distributed systems, adaptation can concern available memory, computation capacity, critical sections, communication bandwidth, power supply of portable devices. They can be interrelated, as e.g., processor frequency and voltage, hence power. An example is a component-based approach, with adaptation mechanisms [22] e.g., for resources disappearing or re-appearing [5].
- Fault tolerance and error recovery also can rely on reconfiguration amongst a redundancy of computation resources e.g., task migration [8] , or degraded modes associated with a management policy.
- Switching of execution mode at application or deployment level in Systems on Chip (SoC) or massively parallel architectures can exploit the availability of different implementations (software or hardware) encapsulated under the same API. An example is in the Gaspard2 framework [24]: modes differ by their characteristics, as quantified by performance evaluation techniques [2], and control mechanisms are defined at specification-level [16, 9] to allow for reconfigurations, implemented e.g., on FPGA [17].
- At application-level, the application-specific sequencing of functionalities, the interactive launching of different tasks dynamically, result in phases with quite different resource requirements.

Some of the adaptations are of a quantitative nature, that can be modeled as a continuously varying system, and controlled by updating parameters [12]. Others are closer to reactive systems with discrete states, between witch transitions are switching upon occurrence of significant events.

Mechanisms of adaptation must concretely support the automatic reconfiguration, on the basis of the classical systems management functionalities, the goal of which is to maintain a system's ability to deliver its service with a prescribed quality [15]. Functions of system management typically include observation and monitoring, configuration and deployment, reconfiguration, resource management, fault tolerance, security. In the framework of the adaptation control loop, the mechanisms of adaptation must be capable of:

sensing or monitoring the state of the system i.e., to monitor and detect causes of reconfiguration; this can involve, in the case of reflexive systems, the maintenance of an explicit model of the state of all concerned entities, at the different levels (hardware, middleware, application, and even possibly environment, as in the case e.g., of fault models for fault tolerance);

acting i.e., they must also have functionalities for installing a new configuration, and deinstalling the former one. They can operate at different levels: task allocation and migration mechanisms, processor frequency control, application-level switch between degraded modes;

deciding : finally, and most importantly, they must have a component to reason upon the past sequences of states and compute the next configuration, in order to close the loop of the adaptive control.

This classical loop is illustrated in Figure 1, reminiscent of the autonomic framework [14]. An autonomic element can be characterized [15] by the application of a control loop, through the management interface, to a managed element. The definition of this management as a control theory problem is seen as one of the research challenges of autonomic systems [13]. The autonomic element can itself be equipped with a management interface, and be reused in the framework of a composite, also regulated in an autonomic way. Autonomy consist of functionalities of self-management: typically self-configuration, self-optimization, self-healing, self-protection [14]. Figure 1 shows how the decision component has to react to incoming events from the platform or environment, taking into account a knowledge on the application and on the platform, and produce actions and reports. We will place our proposal in this framework, in Figure 10.

Approaches to the control of adaptation are explored in different directions, such as rule-based approaches planning, or neural networks. The emphasis on expressivity and dynamicity sometimes leads to systems where the policy defining adaptivity can be changed at run-time. The design of adaptation controllers seems to be often a manual and ad-hoc work, where predictability is not the main issue. However, research progress in architecural definition of adaptive systems have lead to the clear positioning of the closed loop control in the framework of the middleware platforms; it has also begun identifying criteria and objectives for designing the decision component of the control; and it has clarified what are the management actions available for changing the configuration of systems. In this

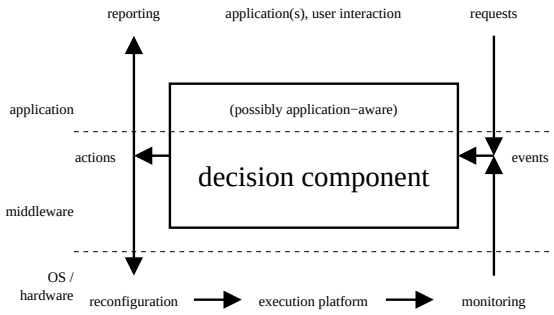

Figure 1: The control of adaptation.

sense, the definition of adaptive systems is mature and ready for reactive and discrete-event systems-based approaches to be considered.

Therefore we claim that a model-based approach, for example based on reactive models and tools, can improve the design of adaptive controllers, for the class of systems where reliability and verifyability are important.

The previous description of adaptation suggests that it resembles control in the broader sense of control theory. Applications of continuous control have been explored e.g., for load management of web servers. We claim here that a parallel approach can be taken fruitfully when considering more logical aspects of systems, where spaces of configurations, execution modes, algorithm versions, have a discrete event system (DES) nature, and are amenable to the application of discrete control notions.

Towards model-based control of adaptation. The proliferation of software, especially in embedded systems, combined with its safety-criticality, plead for the use of well-founded design methods and techniques. The complexity of these systems in size and intricacy makes it essential to have powerful automatic tools for specification, evaluation, verification and transformation, for the sake of the productivity of design.

Formal methods, particularly those aimed at reactive systems, do possess these qualities, and some of them are accompanied by mechanical algorithms for verification and transformation, that are favouring their use by engineers who are experts in the systems to be built, rather than experts in formal models.

Amongst these, Discrete Controller Synthesis (DCS) presents the advantage of being constructive, in the sense that it computes a solution to the problem of the correct control of a system. Indeed, the usability of the approach depends on its understandability, and here the formal technique can be encapsulated as a component of a compilation.

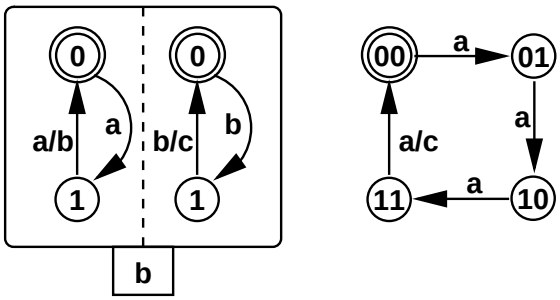

Figure 2: A small example of reactive model.

2.2 Reactive systems and discrete controller synthesis (DCS)

Embedded systems, which are in continuous interaction with their environment, and which have strict safety-criticality constraints, are considered as reactive systems [11]. The synchronous approach to reactive systems [3, 23] provides programmers with concrete tools for design: programming languages, compilers, code generators, model-checkers and also a DCS tool [18].

The model-based approach to reactive systems relies on the basic formalism of labeled transition systems. The finite-state machines are composed of discrete states, that can abstract significant value domains, and characterize significant dynamical sequences of events. Transitions between states are labeled with conditions and actions.

Figure 2 gives a small example of an automaton modeling a reactive system, consisting of a counter of event occurrences modulo four. The left part of the figure shows the parallel composition of two one-bit counters. The first one starts initially from state 0, and upon reception of input a makes a transition to state 1. from there, upon a second occurrence of a, it makes a transition, emitting b, back to 0. It is composed in parallel (with the dotted line, in a surrounding box) with another similar automaton counting two occurrences of b to emit c. The composition defines b to be a local event, in the little attached box.

The right part of Figure 2 shows the automaton, a two-bit counter modulo four, resulting from the synchronous composition, wit input a and output c. We will adopt this graphical syntax, and the synchronous composition semantics, in the following, without detailing formalities here [1].

Such modeling formalisms have been applied to described the control behavior of multi-task systems [19], as illustrated in Figure 3 where a task T_i, initially Idle, receives a request, and depending upon the control signal ok, can go to a waiting state if it is false, or else it can go to the active state Act, which is hierarchical: there it can switch between three modes H, M and L. These multiple modes correspond to different resource management as

well as quality of service policies. The switches are labelled with conditions that control the adaptation.

The synchronous languages [3, 23] (Lustre, Esterel, Signal, Lucid Synchrone, or even Statecharts) provide for high-level language support for the structured construction of such models, enabling the consideration of large and complex systems, the resulting composition being computed by the compilers. As an example, Figure 4, taken from the work on domain-specific language [7], shows the automaton modeling a multi-task application, where grey

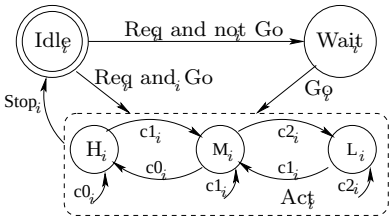

Figure 3: A task model with three modes.

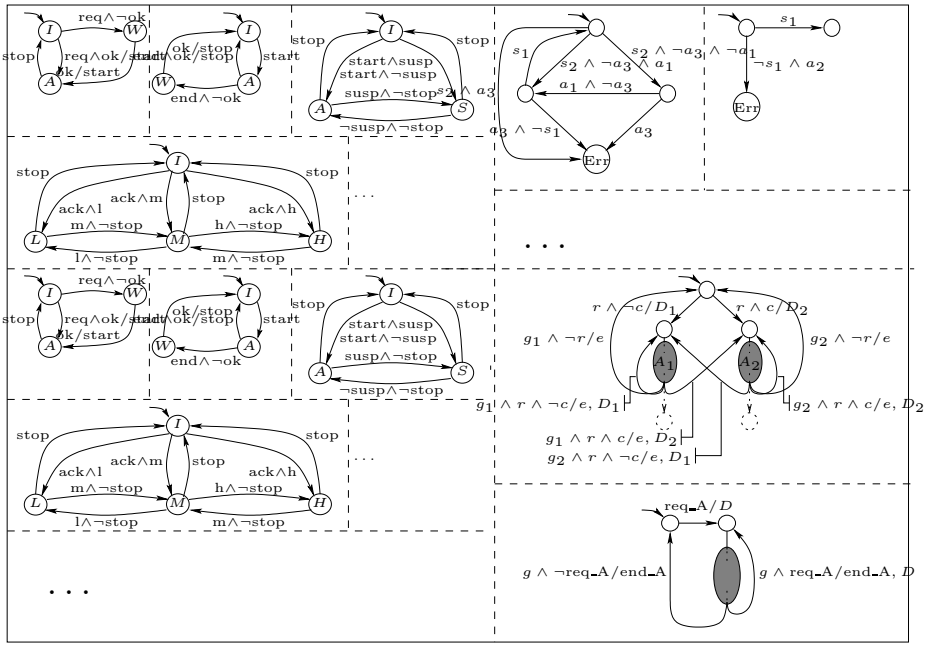

Figure 4: Automaton model of a multi-task application.

zones indicated sub-automata which were not detailed here. One can notice control schemes for putting a task in a waiting state, or mode switching management, resembling Figure 3.

This addresses item 1 of our statement (Sect. 1)

Such automata can also be viewed under an equational form, as a sequential transition function, like in classical Boolean circuits, which is illustrated in Figure 5.

Logical properties of these automata concern typically reachability of states, or invariance of subsets of the state-space, required or forbidden sequences of transitions.

The resulting transition system formal model is the concrete representation of designs upon which operations are defined, in the form of algorithms for their analysis (e.g., for verification or test-case generation) or transformation (e.g., automated partitioning into distributed communicating processes). They constitute executable formal models in the sense that code can be generated for efficient simulation purposes, or for execution, in a platform-dependent way. The underlying models, and their related algorithmic tools, are subject to complexity and combinatorial explosion problems. However, the size of manageable models is such that meaningful systems can take advantage of the offered services of analysis. This is especially true when the models are adequately structured, and the reactive kernel abstracted from the more computational parts of the system.

Expressivity of the models can be augmented with quantitative information (concerning time , but also values of variables that can be handled by static analysis, or even hybrid systems); however, this involves a considerable cost w.r.t. the related algorithms, and the size of manageable systems is therefore limited.

Discrete controller synthesis (DCS) is one of the automated techniques that can be defined to exploit transition system models. It consists of considering on the one hand, the set of possible behaviours of a discrete event system, within which some events are distinguished as being controllable (they can be inhibited), as illustrated in Figure 6.

On the other hand, it requires a specification of a control objective: a property typically concerning reachability or invariance. Examples related to the previous ones are: behaviours should remain within the sub-set of states declared safe, or the termination state of the application should always be reachable (i.e., avoid entering sub-spaces of states from where there is no way to termination).

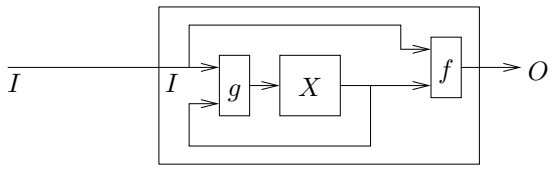

Figure 5: An equational view of a reactive system

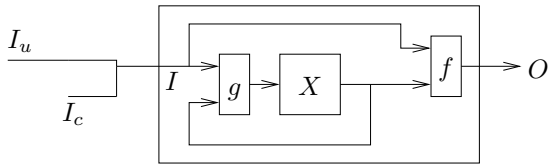

Figure 6: DCS: controllables and uncontrollables

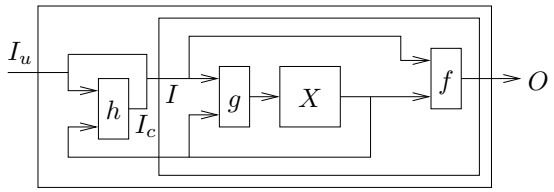

Figure 7: DCS: system under the control of h

This addresses item 2 of our statement (Sect. 1)

DCS consists of the computation of the necessary constraints on controllable events so that the objective properties are satisfied by the resulting controlled system, for all possible uncontrollable inputs. When DCS is maximally permissive, the constraint on controllables is minimal: the most possible behaviors are kept. This can be formulated:

> *Whatever the uncontrollable inputs sequences, the controlled behavior satisfies the objectives.*
>
> *This addresses item 3 of our statement (Sect. 1)*

Figure 7 shows how the controller i.e., the synthesized function h, automatically derived and computed, is such that, in a given state X, given any uncontrollable input I_u, is can give values to controllables I_c, so that the resulting behavior satisfies the control objectives.

Figures 8 shows, from left to right, an example in terms of automata, with five states, and two inputs: one controllable `c`, and one uncontrollable `u`. The objective is to make this system invariant w.r.t. the sub-set of state `E` i.e., to avoid state `s3`. Given the particular transitions, only a smaller sub-set `E'` can be controlled: it is part of the work of the DCS algorithm to explore the state space in order to find the controllable transition where control has to be enforced. the automaton on the right shows the controlled system, where in state `S2` the controllable Boolean `c` is forbidden to take the value true, and must be false, hence inhibiting the wrong behavior to be avoided.

DCS was originally defined in the framework of language theory [21], often called supervisory control of discrete event systems, and is related to game theory. It has been formulated

in terms of labeled transition systems, and involves algorithms that explore the state space in a way resembling that of model-checking verification, symbolically, with complexity issues and capacities of the same order. Within the synchronous approach, DCS has been defined and implemented as a tool integrated with the synchronous languages: SIGALI [18]. It handles transition systems with the multi-event labels typical of the synchronous approach, and features weight functions mechanisms to introduce some quantitative information and perform optimal DCS [8]. We use a tool set is illustrated in Figure 9: models of behaviours are synchronous programs written with the Mode automata language; its compiler can produce a format (called `z3z`) acceptable by SIGALI, which also takes the objectives, and produces a controller. This latter is combined with an executable form of the mode automaton, for co-simulation.

DCS techniques have been studied quite less than verification and model-checking, possibly because they involve a design approach closer to control theory than to computer programming, with a model of the system and its behaviours separately from its control. Applications of DCS have classically concerned discrete control theory problems in manufacturing systems. Computer science applications are less frequent; variants of controller synthesis have been applied on timed automata for application-specific scheduling of task systems, and job-shop scheduling problems.

Despite being less popular than model-checking, it could disseminate at least as broadly, because it offers features of a different usefulness: it is more constructive, in that instead of diagnosing bugs, it produces a solution, which is correct by construction, and finds it, if it exists, even if it is so intricate that a programmer would not have thought of it. Also, its mechanical nature makes it integrable in tools where final users do not need expertise in formal technicities.

Towards DCS-based control of adaptive systems. DCS techniques are meant for the control of reactive systems. They have reached a point of maturity where they are usable as a foundation for model-based control of adaptive and reconfigurable systems. Their continuing evolution and improvement will make their useability even better. They can be integrated

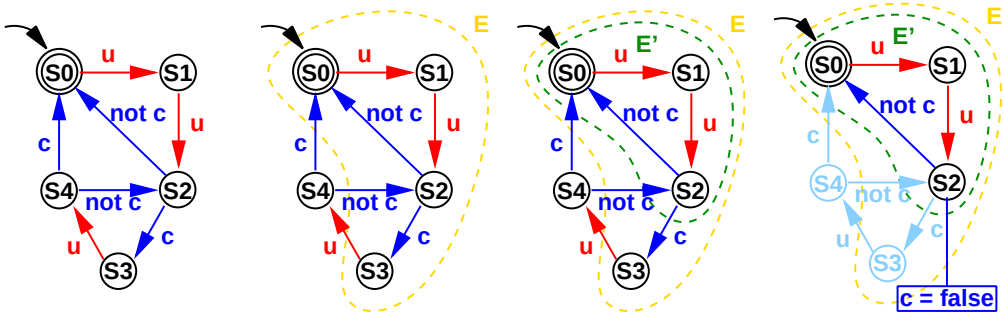

Figure 8: An example of DCS, on an automaton.

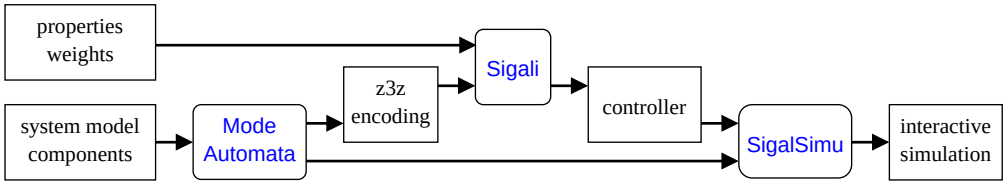

Figure 9: A tool suite for DCS

in a design method that provides users with an automated framework in order to design decision components of adaptive systems, and have a dynamically reconfiguring run-time system platform fulfill the application requirements in a transparent way.

3 Proposals for model-based control

Position. We argue that **model-based control of adaptive systems** can contribute to the goal of **combining adaptivity and predictability** in embedded systems, by defining design methods based on models of the dynamical behaviors involved, and applying automated tools to obtain safe controllers for run-time adaptation. In this position paper, we address these questions for a particular class of systems and properties, expressible as reactive systems and finite-state machines, possibly labeled with some quantitative information. Indeed, at the abstraction level of a reactive kernel managing reconfiguration events and spaces, the adaptation strategy can be formulated as a discrete control problem. DCS techniques can then provide for concrete automated production of correct controllers, that can then be integrated in a run-time platform, connecting this decision component with monitoring and execution facilities.

What: Generating property-enforcing layers.

We propose to contribute to the design, assisted by an automated generation tool, of a middleware-level component or layer for adaptive task and resource control, enforcing the properties defining dynamic reconfiguration strategy.

The controller layer is adaptive w.r.t. the state of the run-time execution platform. It reacts to events concerning the monitoring of resources and activities or requests from the application. It makes transitions between states of configuration, so that the application can be fulfilled, while aspects like fault tolerance, power supply, or communication bandwidth are managed transparently, in an autonomic way.

Using DCS techniques provides us with an automated generation of these controllers. This feature provides for easy modifiability and re-generation of a controller; it opens the possibility for taking into account information about the application, in the form of another transition system model, and generate application-specific solutions, which can make a more optimal use of the platform.

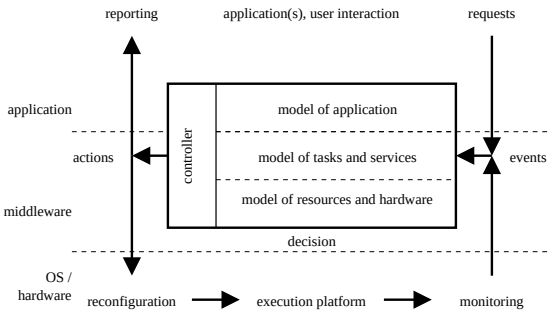

Figure 10: Model-based control of adaptive systems

This addresses item 4 of our statement (Sect. 1)

Figure 10 illustrates our point by showing, in the same framework as Figure 1, how the boxed component at the intermediate layer, at middleware level, interacts with the lower operating system and hardware layer, through monitoring and reconfiguration functionalities, to form a control loop in order to adapt to application requests and other environmental events. This component is built upon models of the platform, the available tasks and services, and also a model of the application, as well as properties defining the control objectives, as seen previously. A controller obtained with DCS constrains the behaviours w.r.t. safety-critical properties. It can also produce reporting information towards the application, such as warning messages, for application-level management.

Why: Safely separating platform control from applications.

The motivation for our work is to alleviate application development from platform details, while allowing for adaptivity in a safe, predictable way. Basing the method upon models and associated algorithms and tools ensures the correctness of the computed controller. The automated tool support enables easy modifyability, and opens the possibility of application-specific solutions.

The encapsulation of the DCS formal tools corresponds to a motivation of making the method user-friendly for developers. Specific languages can provide users with constructs in terms of their expertise, while the compilation cares for the formal technicalities.

This addresses item 5 of our statement (Sect. 1)

How: connecting model-based synchronous tools and reflexive and autonomic middleware. The identification of adaptive middleware mechanisms, and properties to be controlled, lead to the modeling the space of configurations and the operations occurring when making transitions, and of the DCS objectives. The specification of these automata models is made in synchronous languages. The global model can be compiled into a format compatible with the DCS tool, which can be applied with the objectives.

The integration of the resulting controlled system is done in the context of the run-time executive. Events signaling input reconfiguration needs, or output reconfiguration actions, are linked to concrete functionalities of the middleware libraries. The execution of the controller and the reactions of its transition system have to be encapsulated in a reactive component, iteratively executing the controlled transition function.

This addresses item 6 of our statement (Sect. 1)

Preliminary results. Our proposal builds upon previous work, where some of the aspects were partly explored in separate ways. The present position tends towards a generalization, and concretization at middleware-level.

The automatic generation of safe handlers is explored for multi-task systems inspired by robot programming [19, 20], illustrated in Figure 3, in terms of synchronous modeling, featuring notions of resource consumption, and DCS involving some quantitative aspects. This framework is applied to fault tolerance, in the case of fail-silent faults in a multi-processor system [10]. Given a model of the tasks, and a model of the considered faults, a controller automatically migrates tasks, while enforcing essential properties concerning resource usage and the capacity of the processors, as well as functionality fulfillment, whatever the faults occurring. An extension of this works makes usage of optimal DCS in the case of tasks with checkpoints, and the resulting controller optimizes and guarantees adaptivity to the occurrence of fault events, while guaranteeing a predictable response time [8].

A more general framework is then proposed for the automatic generation of property-enforcing layers [1]. The delimitation of the control layer as a relatively small sub-component, at a proper abstraction-level, linked with other, data-oriented code, enables the scalable applicability of DCS. The method offers a separation of concerns between declarative resource constraints and imperative application. It resulted in the tools environment illustrated in Figure 9.

A domain-specific language is proposed upon these bases [7], where multi-task systems can be defined, with resources declared with their sharing properties and capacities, tasks with their various resources consumptions, and different modes between which they can switch. Applications are imperative sequencings of these tasks, with a special alternative operator, to describe different ways of fulfilling some functionality, leaving the choice to the controller to be synthesized. The compilation of this simple language is illustrated in Figure 11. It involves a translation into transition systems, as illustrated in Figure 4, with weights associated with states, and into synthesis objectives. The DCS phase is encapsulated into the compilation process, and hidden to the final user.

In the perspective of autonomic systems, our results can be seen as exhibiting capabilities of self-configuration, in terms of resource usage and task management, of self-optimization in the sense that control takes into account consumptions and quality aspects, and of self-healing in the form of fault recovery control.

Ongoing work is taking place in the direction of implementing the automatically synthesized controllers, in the framework of a control systems programming environment, Orccad [4]. Applications are composed by assembling robot tasks, each equipped by a small activity

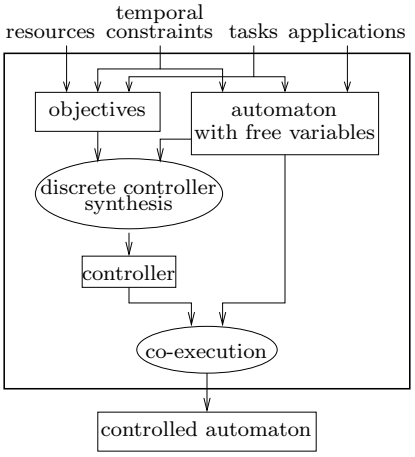

Figure 11: Domain-specific language.

controller modelled as an automaton; the applications are themselves hierarchically controlling sub-tasks through a reactive automaton, in a way that originally inpired our work in [7]. we are modifying models of tasks and applications automata in Orccad, in order to identify controllability, and the resulting controller is designed following the discrete controller synthesis method. The result must then be integrated in the real-time runtime generated by the Orccad environment, where one of the threads is in charge of executing the controller transitions upon the arrival of events. Another direction of work is starting in relation with the Fractal framework [25]. In this component model, components are equipped with controllers, managing typically their life-cycle; composites are themselves components, where the controller manages amongst others the interactions between sub-components. Our goal is to approach the desing of adaptation controllers in this framework by assembling local controllers and adaptation policy properties into a reactive system, and automatically deriving correct controllers.

Opportunities and challenges are examplified by the following research problems and questions raised.

Modeling of adaptation and reconfiguration mechanisms. An important part of the work consists in a modeling activity, where we have to explore what systems do, what we can model properly (i.e., correctly and also efficiently), what we can handle (i.e., what problems we can contribute to).

More precisely, an analysis of actual reflexive and adaptive platforms must be made to determine, while having in view the characteristics of the DCS techniques we use, what are the observation and measure points, what are the control and action points, what are the

properties and control objectives, and how the functional goals and quantitative/qualitative properties complement each other.

DCS techniques. The classical DCS algorithms require to have a global view of the system, which is coherent with a number of adaptive systems, but is not in line with the desire for decentralized control; however, there exist works on modular control techniques that should be explored.

The expressivity of models (and related algorithms) can be extended towards quantitative aspects, like time or value-domains of program variables, or also towards more complex behaviours and properties, involving control with its own state space in addition to that of the controlled system; these extensions have to be approached with consideration of the algorithmic cost that goes with them.

Execution of the controller. Embedding the controller produced by DCS amounts to linking a decision component with the middleware layer, typically with the monitoring functions, and with the task and resource management API. The contents of this component can be optimized by transformation of the synthesized logical constraint in a way that minimizes size or evaluation time.

This controller is the most permissive, and not necessarily deterministic; it has to be completed into a function to be executed. This can be seen as a determinization problem, or one can consider that part of the remaining choices can be done at run-time, on potentially much richer information: typically, quantitative aspects that were abstracted away in the discrete model for the off-line work, can be re-coupled here. The question of having a distributed controller could be approached by using the automated distribution techniques for synchronous programs.

Integration in a user-aimed tool and method. The nature of DCS, at once automatic and constructive, makes it possible to consider its encapsulation into a compilation-like process. There, from a description of resources and tasks in presence, and of an imperative application sequence, the fine-grained, low-level control can be synthesized by a tool, with no involvement from the final user in the possibly intricate synchronizations. This suggest the definition of simple, domain-specific languages, which bring user friendliness by proposing expression in terms familiar to experts of the application fields, and allowing them to remain non-experts of formal methods. In that sense, autonomic and reconfigurable systems are a great opportunity for DCS to be applied as a rigorous approach to control. A challenge coming with this opportunity is that, when no solution is found to the DCS problem, some kind of diagnostic information should be produced, for the designer to modify the design to make it controllable.

4 Conclusion

In this position paper, we argue that **model-based control of adaptive systems**, seen as reactive or discrete-event systems, can contribute to the goal of improving embedded systems with **adaptivity**, while maintaining their safety by continuing ensuring **predictability**. It can do so concretely by performing the off-line, statical, formal computation of run-time,

dynamical, correct controllers. They can handle self-configuration and self-optimization issues of autonomic systems.

It can be done concretely by the modeling of adaption and reconfiguration functionalities at middleware level in terms of reactive systems, the application of discrete controller synthesis (DCS) techniques, taking into account application specificities, the integration of the controller in the run-time executive. The automation of the DCS tools opens the possibility for their encapsulation in a domain-specific language framework, that ensures user-friendliness.

Perspectives are in the modeling of actual concrete middleware platforms, the exploration of more powerful DCS techniques while keeping efficient, the exploration of execution schemes for the generated controllers, and the definition of domain specific languages encapsulating all that.

References

[1] K. Altisen, A. Clodic, F. Maraninchi, and E. Rutten. Using controller-synthesis techniques to build property-enforcing layers. In *Proc. of the European Symp. on Programming, ESOP'03,* April 7 - 11, 2003,Warsaw, Poland, pages 174–188, 2003. LNCS nr. 2618.

[2] R. Ben Atitallah, S. Niar, S. Meftali, and J.-L. Dekeyser. An MPSoC performance estimation framework using transaction level modeling. In *The 13th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, Daegu, Korea, August 2007.

[3] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, January 2003. Special issue on embedded systems.

[4] J.J. Borrelly, E. Coste-ManiÃ¨re, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The orccad architecture. *Int. Journal of Robotics Research, special issue on Integrated Architectures for Robot Control and Programming*, 18(4):338–359, April 1998.

[5] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. A framework for dynamic adaptation of parallel components. In *ParCo 2005*, Málaga, Spain, 13-16 September 2005.

[6] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[7] G. Delaval and E. Rutten. A domain-specific language for multi-task systems, applying discrete controller synthesis. *J. on Embedded Systems*, 2007(84192):17, January 2007. `www.hindawi.com`.

[8] E. Dumitrescu, A. Girault, H. Marchand, and E. Rutten. Optimal discrete controller synthesis for modeling of fault-tolerant distributed systems. In *Proc. of the 1st IFAC Workshop on Dependable Control of Discrete Systems, DCDS'07,* Paris - Cachan, France, June 13-15, 2007, 2007.

[9] A. Gamatié, H. Yu, E. Rutten, P. Boulet, and J.-L. Dekeyser. Synchronous modeling of data intensive applications. Rapport de Recherche 5876, INRIA, April 2006. `www.inria.fr/rrrt/rr-5876.html`.

[10] A. Girault and E. Rutten. Discrete controller synthesis for fault-tolerant distributed systems. In *Proc. of the Ninth Int. Workshop on Formal Methods for Industrial Critical Systems, FMICS 04,* Sep. 20-21, 2004, Linz, Austria, pages ENTCS, Volume 133 , 31 May 2005, Pages 81–100, 2004.
`dx.doi.org/10.1016/j.entcs.2004.08.059`.

[11] D. Harel and A. Pnueli. On the development of reactive systems. In *Logic and Models of Concurrent Systems, NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems*, volume 13, pages 477–498. Springer Verlag, 1985.

[12] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE, 2004.

[13] J. O. Kephart. Research challenges of autonomic computing. In *Proc. of the 27th Int. Conf. on Software Engineering, ICSE'05*, pages 15–22, 2005.

[14] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.

[15] S. Krakowiak. *Middleware Architecture with Patterns and Frameworks*. electronic book, 2007.
`sardes.inrialpes.fr/~krakowia/MW-Book`.

[16] O. Labbani, J.-L.c Dekeyser, P. Boulet, and E. Rutten. Chapter 10: Uml2 profile for modelling controlled data parallel applications. In S. A. Huss, editor, *Advances in Design and Specification Languages for Embedded Systems*. Springer Verlag, September 2007. ISBN: 978-1-4020-6147-9.

[17] S. Le Beux, P. Marquet, and J.-L. Dekeyser. A design flow to map parallel applications onto fpgas. In *17th IEEE International Conference on Field Programmable Logic and Applications*, Amsterdam, Netherlands, August 2007.

[18] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. Synthesis of discrete-event controllers based on the Signal environment. *Discrete Event Dynamic System: Theory and Applications*, 10(4):325–346, October 2000.

[19] H. Marchand and E. Rutten. Managing multi-mode tasks with time cost and quality levels using optimal discrete control synthesis. In *Proceedings of the 14th Euromicro Conf. on Real-Time Systems, ECRTS'02,* June 19th - 21th, 2002, Vienna, Austria, pages 241–248, 2002.

[20] H. Marchand and E. Rutten. Automatic generation of safe handlers for multi-task systems. *Journal of Embedded Computing*, 2007(2), May 2007. *(to appear)*.

[21] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. on Control and Optimization*, 25(1):206–230, January 1987.

[22] M.-T. Segarra and F. André. A framework for dynamic adaptation in wireless environments. In *Technology Object Oriented Languages and Systems Europe (Tools Europe 2000)*, Mont St Michel,France, June 2000.

[23] Synalp. Synchronous applications, languages, and programs, 2008.
`www.inrialpes.fr/synalp`.

[24] DaRT-West team. The Gaspard2 framework for SoC design, 2008.
`www.lifl.fr/west`.

[25] Sardes team and Objectweb. The Fractal Project, 2008.
`fractal.objectweb.org`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399