On augmenting reactivity with deliberation in a controlled manner

Jacek Malec

Department of Computer Science
Lund University
Box 118
221 00 LUND, Sweden
jacek@cs.lth.se

Abstract. We argue that a reactive agent obeying the requirement of predictability imposed by a hard-real-time application domain cannot be equipped with an arbitrarily powerful deliberation capability as this would jeopardize the predictability of the agent's behaviour. Therefore such augmentation should be performed in a principled, controlled manner. We illustrate our line of thought with the example of Generic Layered Architecture used for creating reactive agents acting in dynamic environments requiring real-time responsiveness.

1 Introduction

The interplay between reactivity and deliberation is one of the issues of paramount importance for intelligent agent[1] design. For many years AI research focused on deliberation as the basis for intelligent activity, neglecting the problems of situatedness and reactivity. Increasing complexity of the problems addressed by the researchers has led to the important insight that appropriate reactivity is probably as necessary and important to intelligent behaviour as are the deliberative capabilities. During the last fifteen years the reactive paradigm has gained much attention and popularity, especially in the area of autonomous robotics, and has led to important advancement in the field. However, probably the most important insight is that pure reactivity is much too weak to create any complex behaviours worth the adjective *intelligent*.

The coexistence of the two paradigms of building intelligent agents was not friction-free, but the debate between the proponents of either of the approaches has undobtedly led to better understanding of the problems within the paradigms and also to some solutions to those problems. In the last decade the two approaches began to merge, yielding a number of so called hybrid architectures offering both reactivity to cope with the dynamic environment of the agent and

[1] The term *agent* is understood rather broadly in this paper, encompassing both software agents (softbots living in a networked environment) and hardware agents acting in the real (physical) world, although the main association we would like the reader to have is that of an autonomous intelligent robot.

deliberation to deal with the cognitive tasks. Although the number of proposals is large and covers the whole spectrum between purely reactive and purely cognitive agents, there is no agreement as to how much of both should be used in each particular case.

The balance between reactivity and deliberation is a function of many factors. Among the most obvious ones we can count the designed agent's predominant tasks, or functions, constraining the number of possible implementation architectures, the architectural assumptions of the designer (revealing the designer's preferences), the availability of efficient algorithms for tasks imposed by the application domain (this is especially important for the deliberative part of the system), the necessity to switch between several types of representations used by different parts of the system, etc.

In particular it is important to notice that the space of possible choices is limited by the initial design decisions. Once the architecture is chosen and the representation languages defined the designer is faced with the constraints that have to be obeyed without exceptions. e.g. by choosing ATLANTIS [13] the designer accepts the way the control is imposed over the three layers of this architecture and the temporal dependencies between the layers become fixed. In most cases this is the intended result: the agent architecture is expected to help the programmer achieve maximal functionality in minimum time, relieving her from the tedious task of creating the agent from scratch.

Usually, an architecture assumes a fixed balance between reactivity and deliberation. For example, when using RAPS (Reactive Action Package System, [9]) one decides upon what functionality of an agent is to be implemented at the task (skill) level and what can be explicitly reasoned about at the highest RAP level. However, the price to pay may be some inflexibility of the designed agent. In complex domains one usually wants the possibility of adjusting this balance depending on the current circumstances - thus a reasonable requirement would be to expect the architecture to support such adjustments. In the extreme case one may wish to let the agent decide by itself in a principled manner about this balance, what would imply a meta-reasoning module in the system. Another important limitation imposed by an architecture is the set of representation languages available - architectures are usually centered around some ontological assumptions. The languages used induce a limit on what may be perceived and understood (or reasoned about) by an agent.

Therefore the choice of the languages underlying the used architecture is probably the most important design decision. Before a particular architecture is commited to one needs to carefully analyse its languages and their expressive power (and the associated computational complexity). The languages should allow rich interpretation domains but also efficient reasoning, be it on the reactive level or during deliberation. Moreover, the set of used languages must allow easy translation for effective representation switches.

Yet another dimension of analysis is that of the guaranteed *predictability*[2] of the agent's response. In many applications the agents have to satisfy the usual requirements put on hard-real-time systems. Traditionally AI wasn't concerned much about that issue although it has received some attention during past decades. However, all the applications where an agent might influence the safety of human beings require us to address the problem of predicatbility of the agent's behaviour.

The rest of the paper is organized as follows. In section 2 we introduce an agent architecture called *Generic Layered Architecture* (GLA) and discuss its advantages and disadvantages, in particular for creating intelligent real-time agents. Section 3 introduces the postulate of controlled augmentation of the predictable reactivity with limited deliberation preserving the imposed hard-real-time requirements. Then in section 4 we look at some other agent architectures, focusing mostly on the reactive side of the spectrum. Finally some conclusions are given.

2 Generic Layered Architecture

Our approach to intelligent system design and implementation builds on the three-layered software architecture developed during the last decade at the Linköping University [37] (recently renamed to GLA, the generic layered software architecture). The main distinctive property of this architecture is that it groups similar types of computations into *layers* (shown in Figure 1), as opposed to the much more common approach where a functional decomposition into layers is typical.

Literally dozens of layered architectures have been recently proposed for autonomous system implementation (see e.g. [1, 3, 5, 8, 13] for just a few of them; good overviews can be found in [2, 38, 22]). Although there are differences both in the way of assigning various tasks to different layers and in the way the overall control of the system is executed, the general conclusion is that such layering is beneficial, if not necessary, in designing autonomous *intelligent* systems. We do not claim that the GLA architecture is novel in terms of extending the set of agent's capabilities, rather we expect that the functionalities implementable using GLA might be achieved by using other architectures as well. Our primary claim is that GLA has the following advantages in designing autonomous *real-time* agents:

1. Explicit separation of tasks requiring different conceptual and computational frameworks;
2. Providing a potential designer with a set of formal tools (languages, algorithms) simplifying and systematizing the design process itself;

[2] The term *predictability* is used in this paper in the sense adopted by the real-time community[45]: A system is predictable if and only if all the timing constraints in the system are provably satisfied.

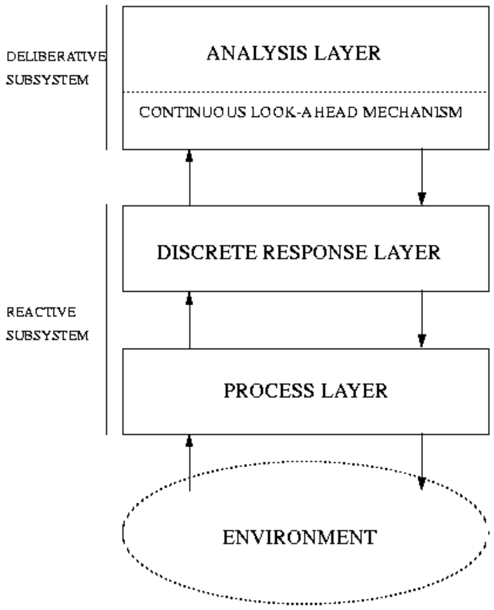

Fig. 1. The generic layered software architecture.

3. Supporting the design process with a set of software tools enabling easy prototyping of complex real-time systems.

The bottom layer, called the *process layer* (PL), is hosting implementations of numerical, periodic tasks, such as identification or control. Data handled by this layer is stored in dual-state vectors. Computations have the form of mappings from input vectors to output vectors and are performed periodically in synchronization with the sample rates of sensors.

The middle layer is called the *discrete response layer* (DRL), and performs tasks which are by nature asynchronous. For instance, it computes the responses to asynchronous events that are recognized by the PL. An example of such a task is the change of control mode (of the PL) due to the change of mode in the environment. The computational model assumed for the DRL is that of discrete event systems (DES). There exist several equivalent DES formalisms: automata, transition systems, rule-based systems, etc. All of them distinguish the notions of *state* and *transition* as central, although the details vary from model to model. We have used the rule-based approach for the purpose of specifying knowledge-based system prototypes. However, this does not preclude usage of other approaches [25, 30]. The process and discrete response layers together form the reactive part of the GLA.

The top layer is called the *analysis layer* because it is defined to handle *symbolic* reasoning tasks such as prediction, planning and scheduling, which require reference to physical time. The output of this layer can be either control events that guide the DRL in its decisions, or parameter settings that are passed through the DRL and directly affect the operation of the PL. This layer has also been tested as a host for symbolic learning mechanisms which adapted the discrete response layer while maintaining its critical response requirements. An important sub-task of this layer, the *continuous look-ahead* (or limited prediction) mechanism has been distinguished in order to identify a predictable subset of the computations in this layer.

While the bottom two layers of GLA, forming the reactive part of the system, might be considered well-developed (both theoretically and in terms of the provided tools, see the description below) and stable, the analysis layer is the subject of on-going research and no design and verification tools have been provided yet.

2.1 Real-Time Software Tools

During our previous research we have thoroughly studied this architecture and its implications on software engineering issues [37]. One of the conclusions was that it facilitates prototyping of systems, especially because it allows development of generic software tools which can be used for implementation of each particular application system. Along this line we have developed software kernels, or *engines*, for development and implementation of the process layer and the discrete response layer. The set of tools includes:

- Process Layer eXecutive (PLX) [34]: a multi-threaded time-triggered real-time engine for implementation of process layer software. It has been implemented on a pSOS$^+$ based system and on a PC running VDX. A port to Real-Time Linux is currently developed.
- Process Layer Configuration Language (PLCL) [35] and its compiler: a language for specification of PL module interconnections and interfaces to both sensors and actuators on one side, and to the DRL software on the other side. The modules themselves are programmed in a subset of some conventional language, such as C or Java.
- Rule Layer eXecutive (RLX) [36]: an engine for implementation of rule-based discrete-event systems in the DRL. It has been implemented on Unix-based machines and on a PC with VDX; A port to Real-Time Linux is under preparation.
- Rule Language (RL) [24]: a rule-based language for declarative specification of discrete-event control.

2.2 The Process Layer Executive

The PLX supports the implementation and maintenance of the hard real-time parts of the application which perform the transformations of periodic data.

During processing, all data is stored in a *dual state vector*, which is a global data structure consisting of an input and an output vector. The values in the input vector represent either sensor readings or internal state, whereas the values in the output vector represent actuator outputs or new internal state.

A PL application defines a sequence of transformations that should be applied to the input vector in order to compute a new output vector. Since sensor values are read periodically, the transformations have to be applied with the same periodicity. When the transformations have been applied, actuator outputs are flushed to devices and the new internal state is installed in the input vector, thus establishing the new state of the process layer.

The PLX engine supervises a PL application, which includes managing the internal state, reading sensors and writing to actuators using user defined access functions, and supervise the execution of the periodical transformations of the dual state vector. The PLX supports decomposition of the vector into several sub-vectors, each of which has its own period, causing transformations to be executed at different rates.

The PLX engine and the PLCL compiler together form the basis for a worst case execution timing (WCET) analysis of a PL application [26, 34].

2.3 The Rule Layer Executive

The rule layer executive supports the implementation of rule-based event processing. In the RLX the state of the world is represented by the time dependent values of a set of symbolic state variables called *slots*. A slot is updated only when its value changes, due to an external event or as a result of a change of another slot's value. Rules specify dependencies between slots, and typically have the following form: *if the value of a particular slot is changed in a certain way, then the value of another slot should also be changed as a result*. Internally, each such change may trigger additional rules, which lead to more updates. This forward chaining process may continue in several steps.

We take an object-oriented view of the rule base in the sense that we associate a set of rules with each slot. This view facilitates the flexibility and maintainability needed for complex systems.

The RLX tool has two major tasks. Firstly, it maintains the set of slots and rules which determines the behaviour of the discrete response layer. Secondly, it performs the forward chaining of rules. The forward chaining is typically initiated by an event recognized by the PL. The result of the forward chaining process may be the change of control algorithm used by some PL process or direct output to a device interfaced by the PL.

The language RL is essentially a syntactic variant of a simple temporal logic and is used for declarative specification of behaviour of the discrete response layer. A comprehensive set of tools for correctness and consistency checking, for timing analysis and for code generation have been developed during the recent years [23, 25, 27].

2.4 An application example

The GLA architecture has been used in recent years for specification and implementation of autonomous agents for several domains requiring hard-real-time guarantees. The original application that triggered our work on GLA was a driver support system [29]. Then we have applied GLA to robot control [31] and recently to RoboCup agent development, to flexible manufacturing cell control [28] and to simulated helicopter control in a traffic surveillance system [24] (an experiment within the WITAS project [7]).

In order to keep this presentation not overwhelmed by unnecessary details, we will briefly present here the first of the applications developed using GLA: the driver support system (DSS). The main function of the system was to provide the driver with accurate, reliable, necessary and sufficient, and timely safety-related support on the basis of unreliable and limited input information, while taking into account the driver's workload and attention capabilities.

The input devices used in our experiments were *autonomous intelligent cruise control* (AICC), *route guidance*, *road/tyre friction estimation system*, and *road-side information system*. However, as the number of intelligent devices available for mounting in a contemporary car becomes larger, there is an obvious need for modularity and incrementality. The reactive part of the GLA can provide at least the second of these properties.

As the support systems mentioned above generate a lot of data, the amount of information potentially available to the driver is very large, and sometimes contains even conflicting messages. In such situation there is an immanent danger of overloading the driver's perceptual and reactive capabilities. Clearly, there is a strong need for systematic handling of the output to the driver from different subsystems in the car, in order to guarantee that the driver gets the relevant information at the right time. This task is complicated by a number of circumstances, for instance:

- Output requests may come from many different sources (support systems) operating independently, or almost independently, without a coherent view of the information available in the whole system.
- Information is time-dependent, so it might prove useless, or even dangerous, if presented to the driver at the wrong time.
- The amount of information the driver can handle is not untimed nor static, but varies dynamically depending on, among other things, the traffic situation, the weather conditions and the kind of maneuvre the driver is performing.
- There may be several output channels available (for example display(s), voice output, and haptic actuators) whose appropriateness may change dynamically.
- Drivers may prefer some output channels to others, or may want to suppress some kinds of messages (e.g. speed excess messages).

The DSS has been specified using the rule language RL introduced in section 2.3. The RL rules are the classical *event-condition-action* rules of the form

```
WHEN event IF condition THEN action
```

where all the events, condition and actions are modeled as value changes to discrete state variables called slots. Some of the actual DSS rules looked as follows:

```
WHEN aicc_distance_acc *= FIRM
  IF aicc_decision |= DISTANCE
  THEN aicc_warning := FIRM;

WHEN acceleration *= ACC
  IF direction_indicator |= LEFT AND car_in_front |= CAR_IN_FRONT
  THEN maneuver := OVERTAKING;

WHEN road_grip *= BAD
  IF skid  |= NO
  THEN road_conditions := BAD;

WHEN on_intersection *= ON_INT
  IF maneuver |= DRIVE_FREELY  OR maneuver |= FOLLOWING
  THEN workload := HIGH;

WHEN skill *= NOVICE
  IF workload |= VERY_HIGH OR light_conditions |= NIGHT OR
     maneuver |= OVERTAKING
  THEN channel := VOICE;

WHEN maneuver *= {FOLLOWING, TURNING}
  IF NOT light_conditions |= NIGHT AND NOT workload |= VERY_HIGH
  THEN channel := DISP_AND_VOICE;

WHEN preference *= NO_SPEED_MESSGS
  IF NOT aicc_state |= AUTO AND NOT aicc_state |= OFF
     AND aicc_decision |= SPEED_LIMIT
  THEN warning := OK;
```

The RL program for the DSS is rather large. It contains 275 rules and 80 slots, where some of the slots have more than 100 possible values. The total number of slot-value pairs is 1752. The program is stratifiable (see [24]) and therefore may be run using the stratified (i.e., simpler) RLX engine. Please note that the specification presented above is *executable*, that is, no further coding is necessary to obtain the implementation for this layer. An automated tool creates also stubs of the PLCL code, to be used in the Process Layer.

The consistency checker gave us guarantee that the DSS program is logically consistent. Then we have performed the timing analysis. Assuming that all the primitive operations take no more than 4μs (implementation-dependent constant), the one-step time bound for this engine was $T_{step} = 0.0108$s. As the maximal number of rules to be fired was 275, the upper bound for a response was $T = 0.88$s, what was deemed acceptable from the application point of view. Some experiments have been performed with computing tighter estimates for the DSS program, according to the proposal described in [27]. However, as the algorithms are NP-complete, the results were expectedly not encouraging - we couldn't obtain any better estimate within 72 hours of computation time due to the complexity of rule dependencies. However, more loosely coupled specifications would theoretically allow the possibility to tighten the estimates.

The Process Layer part of the DSS application was rather simple: it consisted of a number of processes extracting the events from the raw data provided by the input devices and another set of processes translating the discrete decisions into appropriate signals to the output devices (display and voice synthesiser). The timing analysis of this part of the system has provided guarantees that all deadlines for the assumed period of the schedule (100ms) would be met.

The final comment about this application (as well as all others) is that it did not require any advanced deliberation: all necessary decision making could be specified as a set of fixed, effectively propositional, rules describing all modes of the system and all possible transitions between the modes. As we did not posess any possibility of extending our predictability framework onto the analysis layer, we have avoided to face such applications that would require it to be present. In particular, any dynamic cooperation of autonomous (robotic) agents, involving situation analysis and possibly negotiation among agents, would fall into this class and therefore has not been studied so far. The two-robot test case [28] has been created using a simple, fixed communication protocol, and therefore could be realised completely reactively.

3 Controlled deliberation

It is no accident that the tools described in the previous section do not entail the analysis layer. Our approach to the tool generation can be described as bottom-up: we have assumed predictability as the main requirement and then developed and extended the tools in such a way that the created agent can always be subjected to predictability analysis. This way we can formally, declaratively specify an agent's control system using the languages introduced earlier, then subject the specification to correctness analysis, then (mostly) automatically derive (most of) the code and finally analyse its worst-case timing properties. However, the price we pay for this comfortable situation is very high: we can only deal (so far) with purely reactive systems, as the work on development of the tools suitable for creating non-trivial deliberative modules in the analysis layer has lagged much behind the original schedule.

As we have decribed already five years ago in [32], the basic idea behind the controlled deliberation in the analysis layer is to use one of a series of logical formalisms with increasing expressive power and with well-defined computational properties. The agent would dynamically choose the formalism suitable for the environment and the reasoning task it is currently faced with and, possibly, the temporal constraint currently active. The theory of *inhabited dynamical systems* underlying such logical formalisms has been developed by Sandewall [44] where he has introduced a family of logics of action of increasing complexity, suitable for capturing evolution in worlds characterized by some of the assumed set of well-defined criteria, such as temporal invariance of the domain, inertia, alternative results of actions, delayed effects of actions, concurrency of actions, surprises, etc. However, an efficient implementation of such temporal reasoning systems did not appear yet and it seems very unlikely that it will in the near future. On the other hand, one of the most prominent results of this research has recently led to the development of an extremely efficient planning system — the TAL Planner — announced earlier this year [21].

One could imagine the use of TAL as the main tool for reasoning in the analysis layer. Unfortunately, TAL has some peculiarities that preclude its use as a general-purpose mechanism. It requires that for every task it is expected to reason about a specially crafted second-order "control" formula is provided. So far, this formula needs to be created by hand and there are no clues how one could automate the process.

Even if it were possible, the efficiency of TAL, however, is not sufficient to guarantee the success of the whole enterprise — an implementation of the reasoning procedures needs also to be predictable in the hard-real-time sense. The main mechanism used, continuous look-ahead, is based on the same basic principle as Real-Time A* [20], although its complexity is larger due to the fact that each step in predicting the future development of the world involves (in this case) non-monotonic reasoning in a suitable logic. Therefore, in order to provide predictability, one would need to give the worst-case timing estimates for proving theorems in those logics - an problem, if not hopeless, then at least far from being solved.

Given these constraints one can consider several alternative approaches to guaranteeing predictability of the GLA analysis layer. One of them may be the use of interruptible anytime algorithms for deduction. When the alloted reasoning time is over, then conclusions drawn so far are the only ones considered valid. This, of course, Immediately puts the question of validity of this approach: Is it formally correct that some sentences are sometimes false and sometimes true, depending only on the efficiency of the theorem prover? Can it happen that given the same amount of deduction time, the theorem prover will sometimes get an interesting conclusion and sometimes not (maybe depending on the state of the knowledge base or on the proof heuristics used in some particular case)? This raises the need of carefully choosing the granularity of the world description, the proof algorithms used and the time allocation policy. Probably the idea (presented e.g. by Zilberstein and Russell [48]) of using a number of representations in parallel, from coarse to fine-grained, is worth considering in this context.

Another possible line of work consists of pursuing the RTA* approach, as described earlier, but using some appropriately crafted logic for reasoning purposes. Such logic could be e.g. expressed in terms of a labelled deductive system [11], where an appropriate additive algebra on labels would limit the depth of proofs in a theory. However, such label systems would need to be studied in order to establish some correspondence between theories deducible with it and classical theories obtainable using standard logics of AI. The language of the logic itself would also need to be carefully chosen: a rich vocabulary may yield more expressiveness but usually for the price of longer or more complicated proofs. However, in this case such richness might be beneficial, allowing one to use short proofs for non-trivial conclusions. A variation on this theme would be to limit the domain of discourse by considering essentially propositional languages with datalog properties and finite models.

The reason we have not extended the architecture with the deliberation tools earlier should be now obvious: we could not guarantee predictable temporal behaviour of such three-layered architecture. There is still much work to be done before we can announce that GLA fulfills the predictability requirement. In face of the alternatives: a purely reactive system with the necessity of compiling in all the possible reactions, but with guaranteed temporal behaviour, and the full-blown GLA, with rich deliberative capabilities, fast enough in most circumstances but without guarantees, we have to choose the first option as this is the only one that can be used without risk in safety-critical applications.

Of course, the approach described above makes only sense when predictability is accepted as the most important property of the developed system (which it is in case of hard-real-time applications). The important lesson that we have learned during recent years is that if one tries to first use a fast (in terms of development time) but dirty solution, hoping that the details would be fixed later, then the obtained result may happen to be completely useless, forcing the researcher to reconsider the whole theory from scratch. This was what indeed happened with the development of RL (the main formalism used in the middle layer of GLA) and this is what we would like to avoid now. Especially the predictability property has to be embedded in the theory from the very beginning, underlying every algorithm used and every control structure introduced - as we have learnt the hard way, reintroducing it into an existing architecture is virtually impossible.

4 Related work

As mentioned above, a large number of layered agent architectures have been proposed for autonomous agent design and implementation. Chronologically, among the first ones were e.g. NASREM [1], synchronous control of [3], SSS [5], TouringMachines [8], and ATLANTIS [12]. Recently, a number of good overviews has appeared, e.g. chapter six of Arkin's textbook [2] (pointing to the fact that "the nature of the boundary between deliberation and reactive execution is not well understood at this time, leading to somewhat arbitrary architectural decisions", p. 207), Jennings, Sycara and Wooldridge's [18], Müller's [38] or Lee's [22], to mention just a few.

The other three-layered architectures differ from ours in several aspects. Gat [13] distinguishes the layers on the similar basis, i.e. the type of performed computations, but control in his system is located in the middle layer, with only occasional invocation of high-level (actually, path planning) procedures. As in our system, communication with the environment is done through the lowest (control) layer only. The language ALFA is a specialized programming language rather than a formalism for specification of system's abilities.

Connell's system SSS [5] has a similar decomposition of tasks as in our, or Gat's case. The lowest layer hosts control procedures, the middle one contains behavioural specification of system's reactions, and the highest layer is expected to perform path planning. However, in contrast to ours, his approach does not offer any tools for, nor even possibility of, analyzing the real-time performance of the implemented system.

Ferguson's TouringMachine architecture [8] is the most elaborate and complete of the three. It addresses both the real-time aspects and higher cognitive functions (including manipulation of system goals, intentions and beliefs about itself and other agents). The main difference is in the perceptory and effectory paths: TouringMachine layers are simultaneously fed with the sensory information and then compete for the right to control the actuators. Actually, neglecting the possibility of implementing complex continuous control algorithms in our process layer (TouringMachine has totally separated inputs and outputs), the two approaches are to a large extent similar, with our discrete response layer corresponding to the reactive layer of TM, and analysis layer performing the tasks divided between TM's planning and modeling layers.

Another example of a similar, and recently very popular, layered agent architecture is InteRRaP [39]. It can be seen as a more general solution than ours: yet another layer is introduced to provide multi-agent planning and cooperation ability. Another significant difference is that InteRRaP possesses a global knowledge base, accessible from all the processing layers, whereas our approach stresses the need of multiple models suited for the processing layer they are used by. In this way we can have inconsistencies between the models, but on the other hand the modeling is simpler and more realistic. However, the later versions of InteRRaP also allow separated world models, yielding this difference non-existent.

Only a few of the architectures mentioned in this section address the issue of real-time properties. Those that do, e.g., CIRCA [41]; usually adopt the "fast-enough" attitude combined with "anytime algorithms". A more recent architecture, explicitely addressing the hard-real-time requirements, is ARTIC [4]. It is a modification of a blackboard architecture, done in the spirit of GUARDIAN [14], i.e., the hard-real-time response is guaranteed by employing reflexes, while deliberation is performed without any guarantees.

Russell and coworkers have provided a useful theory for anytime algorithms [43, 48], extending the original work by Dean and Boddy on this topic [6]. Thanks to their analysis it is exactly known what can and what cannot be expected from anytime algorithms in the context of real-time applications. The short answer is that soft real-time guarantees can be reasonably derived using probablistic approach, while hard real-time guarantees are much harder to achieve.

An approach ressembling anytime algorithms, but applied to the area of deduction, i.e. the second approach described in previous section, is the one by Fisher and Ghidini [10]. They provide a logical system capable of adapting its deductive power to the resource constraints. However, no useful (for us) bound can be derived for this approach - the only guarantee is that the number of theorems will be smaller in some cases, but the proofs can still be very long and no fixed limit in terms of the number of steps may be given.

The relevant issues of formal analysis of agent architectures has not attracted much attention until very recently. An example of correctness analysis of the classical PRS programs may be found in [46], in its turn based on an early work by Rao and Georgeff [42], while a more general look at the control structures of rule-based systems, relevant from the point of view of our approach, may be found in [16]. However, most literature on this topic needs to be traced in the database theory, where active real-time databases have been studied formally for a while.

Finally, the issue of comparing agent architectures, especially with respect to their effectiveness and suitability for intended applications, is an important topic worth attention and undoubtedly requiring further studies. An interesting preliminary dicussion may be found in [15] and [22].

5 Conclusions

In the paper we have presented the generic layered software architecture (GLA) and commented on the possibility of extending its reactive predictable behaviour with some deliberative capabilities, but in a controlled way. We have described the basic elements of the architecture and some of the tools available (although we have omitted the more advanced tools related to the correctness and timing verification). The we have mentioned Sandewall's theory of inhabited dynamical systems and discussed its usage in extending GLA into a predictable three-layered architecture. Finally we have mentioned a few other possibilities, based on the ideas of RTA* combined with labelled deductive systems.

Our future work on GLA software will be concentrated on guaranteeing the stability of the software engines and the associated tools. We expect to share the current GLA code with the scientific community as soon as a preliminary documentation is available. For more details please have a look at `http://www.cs.lth.se/home/Jacek_Malec/`.

As the last remark regarding the related work one might observe that "intelligent agent" is apparently the buzzword of the last decade. Intelligent software agents, intelligent robotic agents, intelligent real-time agents and intelligent autonomous agents seem ubiquitous in the recent literature in computer science (software engineering and artificial intelligence in particular) and in the systems and control area (especially intelligent control). However, not much attention is being paid to the very notion of an *intelligent agent*: it suffices for a system (either software-, or hardware-based, or both) to reveal a behaviour a bit more complex than just a simple input/output transformation, to be nicknamed intelligent agent. Even better if the behaviour is non-stationary.

What usually hides behind this name is a compound system consisting of several components performing time- and environment-dependent operations over extended periods of time, and composing the results of those operations in some (usually meaningful) way. One may ask though, what is the difference between an intelligent agent and a hierarchical, goal-seeking system, as described already in the 1960s (see e.g. [33]). The answer can be either "none", when it comes to the underlying principles, or "enormous" when we look at the scale of the systems created nowadays. However, the size of the delivered applications should not create an impression that we, agent researchers, are the originators of all the ideas in the field (although we might have rediscovered some of them if we did not read sufficiently much): most of what we discuss today has already been adressed one way or the other in some other area of science. Therefore an interesting research agenda would be to trace the issues underlying agent research back to their roots.

Acknowledgments

The author is indebted to Erik Sandewall and Patrick Doherty for the ideas underlying this paper, to Michael Fisher for a very useful reference and to the anonymous referees for their thorough comments on the earlier drafts of this paper. Probably any useful idea in this paper comes from them, while all the errors and unnecessary complications are, of course, the author's.

Moreover, the author would like to thank Markus for patience.

References

1. J. S. Albus, H. G. McCain, and R. Lumia. NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NIST Tecnical Note 1235, National Institute of Standards and Technology, Robot Systems Division, Center for Manufacturing Engineering, Gaithersburg, MD 20899, 1989.
2. Ronald C. Arkin. *Behaviour-Based Robotics.* MIT Press, 1998.
3. R. Peter Bonasso. Integrating reaction plans and layered competences through synchronous control. In *Proceedings of the Twelvth International Joint Conference on Artificial Intelligence, Sydney*, pages 1225–1231. Morgan Kaufman, 1991.
4. V. Botti, C. Carrascosa, V. Julian, and J. Soler. Modelling agents in hard real-time environments. In *Proc. MAAMAW-99*, pages 63–76. 1999.
5. Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2719–2724, Nice, France, May 1992.
6. T. Dean and M. Boddy. An Analysis of Time-Dependent Planning. *Artificial Intelligence*, pages 49–54, 1988.
7. Patrick Doherty, Gösta Granlund, Krzysztof Kuchciński, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. The WITAS unmanned aerial vehicle project. In *Proceedings ECAI-00*, Berlin, Germany, 2000.

8. Innes A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, University of Cambridge, Computer Laboratory, Cambridge CB2 3QG, England, November 1992.
9. R. James Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Department of Computer Science, Yale University, 1989.
10. Michael Fisher and Chiara Ghidini. Agents playing with dynamic resource bounds. In Markus Hannebauer, editor, *Proc. of the ECAI-00 Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, 2000.
11. Dov Gabbay. *Labelled Deductive Systems, Volume I*. Oxford University Press, 1996.
12. Eran Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Virginia Politechnic Institute and State University, 1991.
13. Eran Gat. Integrating Planning and Reacting in a Heterogenus Asynchronous Architecture for Controlling Real-World Mobile Robots. In *Proceedings AAAI-92*, pages 809–816. AAAI Press, 1992.
14. B. Hayes-Roth. Architectural Foundations for Real-Time Performance in Intelligent Agents. *Journal of Real-Time Systems*, 2:99–125, 1990.
15. Henry Hexmoor, Marcus Huber, Jörg P. Müller, John Pollock, and Donald Steiner. On the evaluation of agent architectures. In Jennings and Lespérance [19], pages 106–116.
16. Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Control structures of rule-based agent languages. In Müller et al. [40], pages 381–396.
17. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufman, 1993.
18. Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.
19. N.R. Jennings and Y. Lespérance, editors. *Intelligent Agents VI, Agent Theories, Architectures and Languages*, volume LNAI 1757 of *LNCS*. Springer, 2000.
20. Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.
21. Jonas Kvarnström, Patrick Doherty, and Patrik Hasslum. Extending TALplanner with concurrency and resources. In *Proceedings ECAI-00*, Berlin, Germany, August 2000.
22. Jaeho Lee. Reactive-system approaches to agent architectures. In Jennings and Lespérance [19], pages 132–146.
23. Man Lin. *Formal Analysis of Reactive Rule-based Programs*. Licentiate thesis, Linköping University, 1997. Linköping Studies in Science and Technology, Thesis No 643.
24. Man Lin. *Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective*. PhD thesis, Department of Computer Science, Linköping University, 1999. Linköping Studies in Science and Technology, Dissertation No 613.
25. Man Lin. Synthesis of control software in a layered architecture from hybrid automata. In F. W. Vaandrager and J. H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 152–164. Springer-Verlag, 1999.
26. Man Lin. Timing analysis of PL programs. In *Proc. 24th IFAC/IFIP Workshop on Real-Time Programming*, Schloss Dagstuhl, Germany, 1999.

27. Man Lin and Jacek Malec. Timing analysis of RL programs. *Control Engineering Practice*, 6:403–408, 1998. Also in: Real-Time Programming 1997, M. Maranzana (ed.), Pergamon Press, 1997.
28. Man Lin and Jacek Malec. Control of a manufacturing cell using a generic layered architecture. In *Proc. Int. Workshop on Robot Motion and Control*, Kiekrz, Poland, 1999.
29. J. Malec, M. Morin, and U. Palmqvist. Driver support in intelligent autonomous cruise control. In *Proceedings of the IEEE Intelligent Vehicles Symposium'94*, pages 160–164, Paris, France, October 1994. IEEE.
30. Jacek Malec. Applied knowledge representation. *CC-AI: The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, 9(1):9–41, 1992.
31. Jacek Malec. On implementing behaviours using a three-layered architecture. In Adam Borkowski and James L. Crowley, editors, *Proceedings of the 2nd International Symposium on Intelligent Robotic Systems*, pages 62–69, Grenoble, France, July 1994.
32. Jacek Malec. A unified approach to intelligent agency. In Wooldridge and Jennings [47], pages 233–244.
33. M. Mesarovic, D. Macko, and Y. Takahara. *Hierarchical Multilevel Systems Theory*. Academic Press, New York, 1970.
34. M. Morin. Predictable cyclic computations in autonomous systems: A computational model and implementation. Licentiate thesis 352, Department of Computer and Information Sciences, Linköping University, 1993.
35. Magnus Morin. PLCL – Process Layer Configuration Language. Technical Report LAIC-IDA-91-TR10, Linköping University, 1991.
36. Magnus Morin. RL: An embedded rule-based system. Technical Report LAIC-IDA-94-TR2, Linköping University, 1994.
37. Magnus Morin, Simin Nadjm-Tehrani, Per Österling, and Erik Sandewall. Real-time hierarchical control. *IEEE Software*, 9(5):51–57, September 1992.
38. Jörg P. Müller. The right agent (architecture) to do the right thing. In Müller et al. [40], pages 211–225.
39. Jörg P. Müller, Markus Pischel, and Michael Thiel. A pragmatic approach to modeling autonomous interacting systems – preliminary report. In Wooldridge and Jennings [47], pages 226–240.
40. J.P. Müller, M.P. Singh, and A.S. Rao, editors. *Intelligent Agents V, Agent Theories, Architectures and Languages*, volume LNAI 1555 of *LNCS*. Springer, 1999.
41. D. J. Musliner. CIRCA: The Cooperative Intelligent Real-Time Control Architecture. PhD Thesis, The University of Michigan, 1993.
42. Anand S. Rao and Michael P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In IJCAI-93 [17], pages 318–324.
43. Stuart J. Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609, 1995.
44. Erik Sandewall. *Features and Fluents*. Oxford University Press, 1994.
45. J.A. Stankovic. Real-time computing systems: The next generation. In *IEEE Tutorial on Hard Real-Time Systems*, pages 14–37. Computer Society Press, 1988.
46. Wayne Wobcke. On the correctness of PRS agent programs. In Jennings and Lespérance [19], pages 42–56.
47. M.J. Wooldridge and N.R. Jennings, editors. *Intelligent Agents, Agent Theories, Architectures and Languages*, volume LNAI 890 of *LNCS*. Springer, 1995.
48. Shlomo Zilberstein and Stuart J. Russell. Anytime sensing, planning and action: A practical model for robot control. In IJCAI-93 [17], pages 1401–1407.