# Intelligent Methods for Embedded Systems

Wilfried Elmenreich[1]

[1]Institute for Computer Engineering,
Technical University of Vienna, Vienna, Austria
wil@vmars.tuwien.ac.at

**Abstract** — *This paper discusses auspicious methods for the implementation of intelligent solutions for embedded systems. An embedded system is a computer system designed to perform a dedicated or narrow range of functions with a minimal user intervention. An intelligent system is a system that is able to react appropriately to changing situations without user input. Main challenges for intelligent solutions in embedded systems come from dependability and real-time requirements and from constraints on cost, size, and power consumption.*
*Possible intelligent methods for embedded systems are biologically inspired, such as neural networks and genetic algorithms. Multi-agent systems are also prospective for an application for non-time critical services of embedded systems. Another field is soft computing which allows a sophisticated modeling of imprecise (sensory) data. Finally, since embedded systems often provide critical services, there is need for intelligent validation techniques that assist the developer in evaluating if the system is fit for its purpose.*

## 1 Introduction

Designing embedded systems is quite different from desktop programming. Desktop programmers are able to use standard environments with almost unlimited (virtual) memory as well as a good monitoring and debugging interface. In contrast, an embedded programmer has to renounce such comfort. He or she has to cope with many different microcontrollers, some of them providing only several kBytes of program memory and a few bytes of working memory. Instead of a comfortable environment with graphical screen display and printers, developers have to use LEDs, the display of an oscilloscope, or a serial data stream for debugging. Embedded systems should run on sparse resources, thus should require low power, little program and working memory, be small in size, and often should guarantee real-time behavior or be resistant against failures. Many embedded applications have been implemented as “dumb” programs, consisting only of a few lines of code. However, the embedded market calls for more extensive and smarter applications, for example, an electronic braking system of a car should be able to provide its service in various extreme situations, e. g., the breakdown of a braking element of one wheel.

Therefore, a new generation of *intelligent* embedded systems is necessary. Most algorithms for *intelligent* embedded systems already exist in the computer science community,

however, due to the fundamental difference between desktop and embedded computing, many approaches need to be reviewed in order to determine if they are applicable for embedded systems.

It is the objective of this paper to provide an overview on intelligent methods that are prospective for miscellaneous tasks of embedded applications. The remaining parts of the paper are structured as follows: Section 2 provides the necessary definitions for intelligent systems and discusses some misconceptions around that term. Section 3 lists a number of possible benefits for intelligent solutions in embedded systems. Section 4 explains, why biologically inspired systems are a good template for intelligent embedded systems. Section 5 explains the concept of multi agent systems and refers to some applications of multi agent systems in the embedded domain. Section 6 motivates and proposes the concept of soft computing for embedded systems. Section 7 discusses the methods of model checking and fault injection for system validation. The paper is concluded in Section 8.

## 2 Definition of Intelligent Systems

"Intelligence" refers to the overall effectiveness of an individual's mental processes, particularly his or her comprehension, learning/recall, and reasoning capacities. When intelligence is seen as the capability to solve (new) problems, it is possible to identify "intelligent" solutions in engineering. It should be made clear, that "intelligent behavior" in this context does not refer to the ability to solve puzzles or to be conscious of ourselves being. These goals of a strong artificial intelligence are far above the capabilities of our current machines and computers. Intelligence in engineering means systems that are able to react appropriately to changing situations without input from a human operator. In other words, an intelligent algorithm is one that is able to solve problems that stem from changing situations. This definition, of course, encompasses a wide range of engineering applications and many different methods and algorithms. The research on intelligent systems is motivated by the high versatility of such systems, which makes it possible to reuse many algorithms successfully in different applications.

There are some misconceptions around intelligent systems. For example, a system that does not have intelligent behavior on its own is probably nevertheless the product of an intelligent mind, a fact that might be used by the advertising department to call the product itself intelligent.

Moreover, an intelligent system might not be the best solution for a given application case. For example, a racing bicycle is specifically designed for good roads, while the human legs adapt well to various terrains like grassland, water, mountains, etc. Therefore, the human locomotor system represents an intelligent system while a racing bicycle is none. Nevertheless, it is probably advantageous to choose the bike in some cases.[1]

At last, an intelligent system must not be necessarily complex. Leslie Smith [1] describes the example of a robot that navigates towards a light source. Primitive biological systems tend to solve these types of tasks in a simple, but effective, way.

[1] Note that the combination "human with bicycle" represents an intelligent system, when one is free to choose to use the bicycle or not.

## 3 Motivation for Intelligent Embedded Systems

Naturally, the word "intelligent" (as well as smart, wise, clever) transports a very positive meaning, however, it is obvious that it should be analyzed why using intelligent solutions for embedded systems is advantageous.

The following potential reasons for employing an intelligent solution can be identified:

**Dependability:** Applications for harsh environments such as process control applications call for a solution that adapts to changing situations like performance loss or breakdown of a component. For such applications, intelligent solutions enable graceful degradation or self-stability properties.

**Efficiency:** An intelligent solution might be able to increase efficiency of the given resources.

**Autonomy:** An intelligent solution might be able to perform the same task as a traditional system without or with reduced requirement for human supervision or interaction.

**Easy Modeling:** An intelligent generic self-organizing solution liberates the system designer from modeling and implementation issues. This reduces the chance of human error and reduces cost and time in the design phase.

**Maintenance costs:** An intelligent system might require less frequent service iterations since it is able to run for long durations without human interaction.

**Insufficient alternatives:** Sometimes there is no traditional approach to solve a given problem satisfyingly, which forces the application of an intelligent solution. For example in data analysis, the application of neural networks solves the problem of nonlinear correlations, which is not supported by traditional approaches [2].

Note, however, that the modelling issue gives also reason for a counter argument against intelligent solutions. While a traditionally designed system usually *must* be understood by the designer, this can be different for particular intelligent solutions. For example, while neurons are well understandable, their application in a neural network leads to an emergent system which cannot be fully described by a simple model. If such a system should be used in a critical application, problems for the dependability analysis and certifications arise.

As a positive, almost all above described potential advantages support also cost reduction. Such cost savings can appear at the design time of a system or during maintaining the system.

## 4 Biologically Inspired Embedded Systems

Nature has shown to be a great inventor of intelligent solutions in embedded systems. The reasons for that fact are given by two major requirements that are put on biological systems:

First of all, most biological systems have to work autonomously. Usually, there is no one to help a creature in recovering from a breakdown. Therefore, animals, as well as plants, had to develop strong methods of self-healing and automatic recovery. Of course, there is positive interaction between individuals: Mammals protect their new-generation,

many animals live or hunt in groups, and there is the concept of symbiosis that involves even multiple different species. However, when regarding any of these groups of interdepending individuals, a system of strong inner connection that is able to help itself can be made out.

The second requirement that biological systems have in common with embedded engineering systems is that most components are employed to perform a dedicated or narrow range of functions. Such biologically embedded systems are always optimized to the same goals as embedded computer systems: creation effort, maintenance effort, size, weight, power consumption.

### 4.1 Neural Networks

The most frequent example for biologically inspired computing is that of neural networks (NNs). A NN consists of interconnected neurons, each with a set of input and output connections. In principle, a neuron contains a simple add-and-compare mechanism that sums up the input signals and generates an output signal (i. e., the neuron "fires") if a particular threshold has been exceeded. While the concept of such a neuron cell is very simple, a whole NN shows emergent properties such as learning and reasoning (for example, think of the abilities of the human brain, a NN with about $10^{10}$ neurons). NNs are extreme versatile. According to the theorem of Hecht-Nilsson [3], *any* given function can be expressed by a three-layer NN with an appropriate number of neurons.

An impressive feature of a NN is the ability to *learn*, which enables such systems to adapt to changing conditions [4]. NNs support supervised and unsupervised learning. In supervised learning, back-propagation NNs are used. During a training phase, the parameters of the NN are adapted until the system performs the desired function. The trained system is then able to perform the programmed function with a high robustness against errors. An example for such an application in the embedded domain is given in [5], where an artificial NN is used to filter out errors from infrared distance sensors.

Unsupervised learning algorithms try to extract common sets of features from the input data [4]. An example for an unsupervised learning artificial NN is Kohonen's self-organizing map [6]. Unsupervised learning algorithms are used for automatic classification, modeling, and data compression systems.

Drawbacks of NNs are its black-box data processing structure and, in some cases, a slow convergence speed. Thus, the data processing mechanism of a NN cannot be programmed, understood, or verified in terms of rules.

### 4.2 Genetic Algorithms

A genetic algorithm (GA) is a derivative-free and stochastic optimization method that builds on ideas from the natural selection and the evolutionary process [7]. It is some kind of search algorithm that is advantageous if the given search space is too large to be searched by exhaustive search algorithms and too unstructured to be able to use straight forward search algorithms. Moreover, a GA needs only a minimum on information about the problem to be solved and is thus easily applied.

Basically, a GA needs an initial population of "genes", an algorithm that allows to cross-mix these genes, and a fitness function that produces a comparable value on the quality of an actual solution. After recombination and mutation of genes the GA uses the fitness

function to select the best genes for the new population. By making multiple iterations, the GA approaches an solution that is equal or better than the start value.

An example for the application of a GA, which is relevant for embedded systems, is given by Atanassov in [8]. The work focuses on the search for an input set with the maximum (worst case) execution time (WCET) of a given program on a given execution environment. Knowledge of the maximum execution time of a program is essential for several hard real-time architectures, such as the Time-Triggered Architecture [9]. The search for the WCET input set is non-trivial due to the usually large input space and the inhomogeneity of the search space due to code dependencies and side effects. Atanassov's approach estimates the WCET of a program by measuring the execution time with several input sets and then modifying the input sets towards the maximum execution time. He uses a GA for finding input sets with large WCETs, whereas the fitness function is given by the execution time with the respective input set. The evaluation of this approach shows that the genetic algorithm is able to find a rather good solution in relative short time (Atanassov's experiments ran for several days on an embedded C167 processor), however, it reveals also an inherent drawback of GA – since the search is not exhaustive, the algorithm can get stuck in local extrema, thus fails to find a global optimum of the fitness function. In other words, GAs are usually very fast in finding a good solution, but in general they will not find the best solution.

### 4.3 Neuro-Fuzzy Systems

Fuzzy Logic forms a bridge between digital rules (for example "*if measured flow is greater than 50* $\frac{l}{s}$*, then open valve") and imprecise information (for example* "flow is between 48 and 52 $\frac{l}{s}$"). The inference method of Fuzzy logic is similar to the human brain. Fuzzy Logic supports the implementation of control algorithms for imprecise sensors that perform better than traditional control methods. An exhaustive introduction into Fuzzy Logic with a focus on control methods can be found in [10].

However, Fuzzy Logic has the drawback of lacking an effective learning mechanism – auto-tuning a classical Fuzzy system is difficult [7]. The combination of Fuzzy systems with neural networks overcomes some problems of NNs and Fuzzy Logic, by providing an adapting system with a rule-based model. Such neuro-fuzzy Systems employ learning algorithms of a NN to determine the parameters of a Fuzzy inference system.

Unlike NNs, a neuro-fuzzy system is always interpretable in terms of fuzzy *if-then* rules, thus giving insight into the model.

## 5 Multi-agent Systems

Wooldridge and Jennings define a multi-agent system (MAS) as a hardware or (more usually) software-based computer system that provides the following properties [11]:

**Autonomy:** *Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.*

**Social ability:** *Agents interact with other agents (and possibly humans) via some kind of agent-communication language.*

**Reactivity:** *Agents perceive their environment, (which may be the physical world, a user*

*via a graphical user interface, a collection of other agents, the internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it.*

**Pro-activeness:** *Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.*

The idea of a multi-agent system (MAS) is to interconnect several widely independent agents, thus enabling this ensemble to function beyond the capabilities of a single agent of the set-up [12]. In general, MASs may enhance speed (due to parallelism), reliability (due to redundancy), efficiency, and flexibility. A frequent paradigm for an MAS is an automated travel agency that uses the internet to engage other agents, to perform flight reservations, etc. Recent research has also shown the applicability of MAS to the embedded systems domain. An example is the PABADIS project [13], which employs an MAS in a fieldbus system for factory automation [14]. There are, however, some critical problems for MAS:

**Communication:** Agents must agree on a common transport protocol and a common communication language in order to interact properly with each other [12]. For example, an MAS that spans different embedded systems probably will have to deal with differing data representations and semantics.

**Integration of existing applications:** Especially in the embedded systems domain many applications come in the form of legacy applications, i. e., systems that have been designed according to their own rules and conventions [15]. Since the participation of such systems was not in their designers' mind, there can be remarkable effort in implementing an appropriate interface for MAS.

**Real-Time Capabilities:** Due to the loosely coupling of the single agents, agents of an MAS are typically asynchronous and, therefore, prone to race conditions, temporal unpredictability, and, in the worst case, to deadlock situations.

**Supervision:** It is often difficult to monitor the behavior of every single agent with respect to real time. Monitoring tasks can change the behavior of an MAS due to the so-called probe-effect [16].

Despite these problems, MAS are apt to implement intelligent functions for embedded systems. Due to the problems in temporal predictability, applications for MAS lie mainly in non time-critical applications, such as, for example, configuration tasks.

## 6 Soft computing

The world of computers are digital, and at a very low level, only 0 or 1 exist (or *true* and *false*). The properties in the real world, on the other hand, are often different from that black-and-white thinking, which often is the reason for problems in embedded systems, where a system views its components either as correct (according to its specified service) or incorrect (outside the service specification). So, a component that provides its service only a little bit offside its specification might be perceived as correct or incorrect by a correct system, which causes a divergence in the correct system states.

To overcome such real life complexities such as imprecision the computer paradigm of *soft computing* is used. Soft computing is an overall term for a coalition of methodologies

such as Fuzzy Logic, neuro-computing, evolutionary computing, probabilistic computing, chaotic computing and machine learning [17]. Neural Networks, Genetic Algorithms, and Neuro-Fuzzy systems, which have been already discussed in this paper in the context of biologically-inspired computing can be also seen as a kind of soft computing. Since many soft-computing approaches are also biologically inspired, there is a great intersection of these two disciplines.

Apart from the digitalization problem, sensor measurements get an additional dimension by regarding their accuracy in the value and time domain, and, since sensors can fail, the reliability of a particular measurement.

Embedded systems often contain sensors that form the borderline between the digital computer world and its analog environment. Instead of reducing a sensor measurement to its value, it is often advantageous to add some additional information about the probability of the given value. Buede and Waltz discuss the benefits of such soft sensors and probabilistic sensor fusion in [18]. An architecture for sensor fusion with probabilistic measurements can be found in [19], where each sensor measurement is attributed with a confidence value that indicates the accuracy of the measurement.

## 7 Model Checking

Usually, intelligent solutions lead to complex systems, where it is often difficult to prove the correctness of the presented solution. For example, a neural network consists of several neural nodes called neurons, where each neuron acts deterministically according to its input. However, when the neural network is regarded as a whole, it is difficult to prove a particular behavior. Thus, there is a need for intelligent methods that assess if a given system is fit for its purpose.

A possible verifying method is *model checking*, a technique for verifying finite state concurrent systems. The system-under-test is modelled by a set of state variables and the possible transitions that can take place between the particular states. A set of properties distinguishes the intended (allowed) states from the unintended states. A model checking tool will then automatically search the reachable state space in order to verify that only intended states can be reached.

The main disadvantage of this approach is the state explosion problem that occurs when verifying systems with high parallelism or large data domains. Therefore, the state explosion problem has been target of research which led to some remedies for the problem, to name a few: symbolic model checking [20], abstraction [21], symmetry [22], and induction [23].

In contrast to traditional approaches, such as simulation, testing, and inductive reasoning, model checking is quite fast in detecting subtle bugs for several applications [24]. As an advantage, the model checking process always presents a counterexample if a specified property does not hold for a given design. After setting up the model, the checking algorithm is fully automatic and thus requires no user supervision for its application, which makes it an intelligent tool. Due to recent research, model checking is more and more used for embedded systems [25].

| Method | Characteristics | Sample embedded applications |
|---|---|---|
| Artificial Neural Networks | unsupervised or supervised (trained) learning, black-box data processing structure | signal calibration [5], pattern recognition [], classification [7] |
| Genetic Algorithms | optimization method especially for large and unstructured search space | real-time scheduling [26], program timing analysis [8] |
| Model Checking | automated method for verifying finite state concurrent systems | verifying properties of distributed systems [25] |
| Multi-Agent Systems | distributed system of self-acting programs acting in a network | flexible automated manufacturing [14] |
| Neural Fuzzy Systems | integration of Fuzzy inference rules with neural networks | intelligent control [27], embedded expert systems [] |

Table 1: Overview on presented methods

## 8 Summary and Conclusion

All presented methods, summarized in Table 1, can be potentially used for the conception, design and utilization of intelligent systems for particular embedded applications. There is no all-round solution for an intelligent solution – it will be necessary to evaluate from case to case if an approach is propitious or not. Furthermore, we should keep in mind that it is possible to use most methods in a synergistic way, which may lead to the most favorable solution in some cases.

Some existing disadvantages, such as the resource requirements on memory and computation for implementing a neural network can be overcome by applying an appropriate hardware-software co-design which is a common approach for many embedded designs.

However, a major problem for many intelligent solutions is that they come in the form of a complex system, which cannot be easily evaluated analytically – it is often difficult to check whether a system is fit for its purpose or not. This problem calls for intelligent validation methods like model checking.

## Acknowledgments

I would like to thank my colleagues Wolfgang Haidinger and Martin Schlager for giving me inputs to this paper. This work was supported by the Austrian Ministry of Science via project CoMa (H-965/2002).

## References

[1] L. S. Smith. *Biologically-Inspired Systems*. Department of Computing Science and Mathematics, University of Stirling, Scotland, Scotland, UK, 1999. Lecture notes on Biologically Inspired Computing.

[2] E. C. Malthouse. Limitations of nonlinear PCA as performed with generic neural networks. *IEEE Transactions on Neural Networks*, 9(1):165–173, January 1998.

[3] N. Nilsson. *Learning Machines*. Morgan Kaufmann, San Mateo, 1965.

[4] D. D. Lee and H. S. Seung. Learning in intelligent embedded systems. In *Proceedings of the Workshop on Embedded Systems*, Cambridge, MA, USA, March 1999.

[5] H. Kraut. Signalverarbeitung mittels eines Neuronalen Netzwerkes für einen Smart Sensor. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2003.

[6] T. Kohonen. *Self-Organizing Maps*. Springer, Heidelberg, 1995.

[7] X. Z. Gao and S. J. Ovaska. Soft computing methods in motor fault dianosis. *Applied Soft Computing*, 1:73–81, 2001.

[8] P. Atanassov, S. Haberl, and P. Puschner. Heuristic worst-case execution time analysis. In *Proceedings of the 10th European Workshop on Dependable Computing*, pages 109–114. Austrian Computer Society (OCG), May 1999.

[9] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple. Time-Triggered Architecture (TTA). *Advances in Information Technologies: The Business Challenge, IOS Press*, 1997.

[10] W. Pedrycz. *Fuzzy Control and Fuzzy Systems*. John Wiley & Sons, 1993.

[11] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[12] H. S. Nwana and D. T. Ndumu. A perspective on software agents research. *Knowledge Engineering Review*, 14(2):1–18, 1999.

[13] The PABADIS consortium. *Revolutionising Plant Automation The PABADIS Approach*, 2002. Available as white paper at http://www.pabadis.org.

[14] Y. K. Penya, S. Mahlknecht, and P. Rössler. Lightweight agent platform for high-performance fieldbus nodes. In *Proceedings of the Work-in-Progress Session of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'02)*, page 71, Västerås, Sweden, August 2002.

[15] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585) Deliverable CSDA1*, October 2002. Available as Research Report 54/2002 at http://www.vmars.tuwien.ac.at.

[16] C. E. McDowell and D. P. Helmbold. Debugging concurrent programs. *ACM Computing Surveys*, 21(4):593–622, December 1989.

[17] L. A. Zadeh. Applied soft computing – foreword. *Applied Soft Computing*, 1:1–2, 2001.

[18] D. M. Buede and E. L. Waltz. Benefits of soft sensors and probabilistic fusion. *Signal and Data Processing of Small Targets, Proceedings of the SPIE*, 1096:309–320, 1989.

[19] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[20] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

[21] E. M. Clarke, Jr., O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[22] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1995.

[23] K. S. Namjoshi. *Ameliorating the State Space Explosion Problem*. PhD thesis, University of Texas at Austin, 1998.

[24] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[25] S. Gnesi. Model checking of embedded systems. *ERCIM News*, (52), March 2003.

[26] M. T. Jensen. Improve robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing*, 1:35–52, 2001.

[27] A. Abraham and B. Nath. Designing optimal neuro-fuzzy systems for intelligent control. In *Proceedings of the 6th International Conference on Control, Automation, Robotics, and Vision*, December 2000.