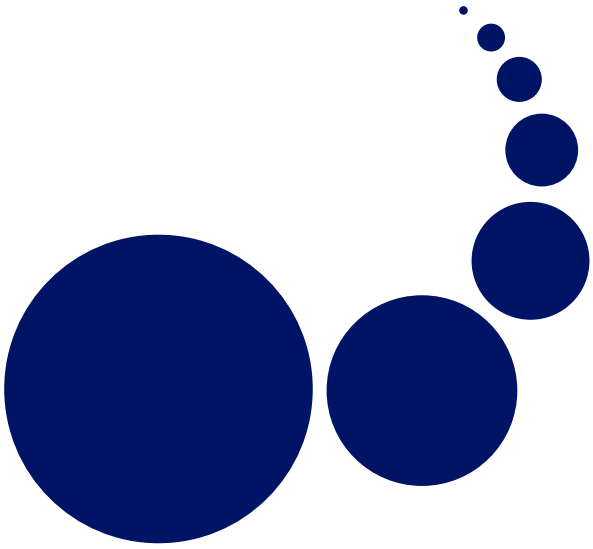
SCALABLE COMPUTING
Practice and Experience

Special Issue: Real-Time Distributed Systems and Networks

Editor: Janusz Zalewski



Volume 10, Number 3, September 2009

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 10, Number 3, September 2009

TABLE OF CONTENTS

Special Issue Papers:

Scalable Computing: Practice and Experience
Volume 10, Number 3, p. i. http://www.scpe.org
ISSN 1895-1767

INTRODUCTION TO THE SPECIAL ISSUE: REAL-TIME DISTRIBUTED SYSTEMS AND NETWORKS

1. Introduction. The contents of this Special Issue is composed of extended versions of eight papers presented at the 2008 edition of the Real-Time Software Workshop (RTS´08), held in October 2008, in Wisła, Poland, as a part of the IMCSIT 2008 (International Multi-conference on Computer Science and Information Technology). The papers included in this issue all focus on Real-Time Distributed Systems and Networks. Papers from other focus areas of the RTS´08 workshop were submitted for publication elsewhere, and some od them have recently appeared [1].

2. Contents of This Issue. The area of real-time distributed systems and networks is very broad, which is reflected by the types of papers included here. However, the papers can be categorized based on the systems development perspective. In this view, one can group papers in three essential categories, regarding whether they deal with the methods, techniques, or tools for system development.

Papers in the methods category vary from more theoretical to practical aspects of distributed system design and development. At the theoretical end, Le Berre et al. present a state-based modeling approach to analyzing distributed embedded systems, based on the Temporal Logic of Actions, TLA+. More practical approaches are presented in two other papers. Galdun et al. discuss a method for increasing reliability in networked control systems via redundancy, and demonstrate it in a case study of a 4-rotor helicopter. Gonnord and Babau, in turn, discuss resource management in embedded systems, by handling resource constraints using quality-of-service criteria, and demonstrate it for an image processing example using their framework named Qinna.

Furthermore, Moritz et al. propose a method for handling real-time capable embedded web services, and apply it in a mobile robot case study, while Binotto et al. offer a new method of dynamic task reconfiguration, in an aspect oriented framework, and apply it in a UAV based surveillance system.

Regarding more specific techniques for real-time distributed system development, two papers are included here. Martinez et al. give an application perspective on two case studies in wireless sensor networks: a healthcare system and a networked control system. Finally, Song discusses a variety of localization techniques for mobile sensor networks. The only paper related to tools, by Rzońca et al., discussing a development environment for programming small controllers, based on the IEC Std 61131-3, concludes this special issue.

3. Conclusion. In this Editor´s opinion, the eight papers presented here provide a good, although selective, overview of the subject area, and offer an interesting perspective on problems and related solutions in real-time distributed systems and networks.

For preparation of the RTS´08 Workshop thanks are due to the Workshop Program Committee co-chairs, Professors Wojciech Grega from AGH University of Science and Technology, in Kraków, Poland, and Andrew Kornecki from Embry-Riddle Aeronautical University, in Daytona Beach, Florida, USA.

Janusz Zalewski,
SCPE Editorial Board Member and RTS'08 Workshop Co-Chair,
Dept. of Computer Science,
Florida Gulf Coast University,
Fort Myers, FL 33965,
USA,
zalewski@fgcu.edu

REFERENCES

[1] J. Zalewski, Guest Editor, *Special issue on real-time safety-critical systems*, Innovations in Systems and Software Engineering, 5(2), pp. 95–161 (June 2009).

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 229–239. http://www.scpe.org
ISSN 1895-1767

REAL TIME BEHAVIOR OF DATA IN DISTRIBUTED EMBEDDED SYSTEMS*

TANGUY LE BERRE, PHILIPPE MAURAN, GÉRARD PADIOU, PHILIPPE QUÉINNEC†

Abstract. Nowadays, embedded systems appear more and more as distributed systems structured as a set of communicating components. Therefore, they show a less deterministic global behavior than centralized systems and their design and analysis must address both computation and communication scheduling in more complex configurations. We propose a modeling framework centered on data. More precisely, the interactions between the data located in components are expressed in terms of a so-called observation relation. This abstraction is a relation between the values taken by two variables, a source and an image, where the image gets past values of the source. We extend this abstraction with time constraints in order to specify and analyze the availability of timely sound values.

The formal description of the observation-based computation model is stated using the formalism of transition systems, where real time is handled as a dedicated variable. As a first result, this approach allows to focus on specifying time constraints attached to data and to postpone task and communication scheduling matters. At this level of abstraction, the designer has to specify time properties about the timeline of data such as their freshness, stability, latency... As a second result, a verification of the global consistency of the specified system can be automatically performed. The verification process can start either from the timed properties (e.g. the period) of data inputs or from the timed requirements of data outputs (e.g. the latency). Lastly, communication protocols and task scheduling strategies can be derived as a refinement towards an actual implementation.

Key words: real time data, distributed systems, verification

1. Introduction. Distributed Real Time Embedded (DRE) systems are increasingly widespread and complex. In this context, we propose a modeling framework centered on data to specify and analyze the real time behavior of these DRE systems. More precisely, such systems are structured as time-triggered communicating components. Instead of focusing on the specification and verification of time constraints upon computations structured as a set of tasks, we choose to consider data interactions between components. These interactions are expressed in terms of an abstraction called *observation*, which aims at expressing the impossibility for a site to maintain an instant knowledge of other sites. In this paper, we extend this observation with time constraints limiting the time shift induced by distribution. Starting from this modeling framework, the specification and verification of real time data behaviors can be carried out.

In a first step, we outline some related works which have adopted similar approaches but in different contexts and/or different formal frameworks.

Then, we describe the underlying formal system used to develop our distributed real time computation model, namely state transition systems. In this formal framework, we define a dedicated relation called observation to describe data interactions. An observation relation describes an invariant property between so-called *source* and *image* variables. Informally, at any execution point, the history of the image variable is a sub-history of the source variable. Actually, the source is an arbitrary state expression. An observation abstracts the relation between the inputs and the outputs of a communication protocol or between the arguments and the results of a computation.

To express timed properties on the variables and their relation, we extend the framework so as to be able to describe the *timeline* of state variables. Therefore, for each state variable x, its timeline, an abstraction of its time behavior, is introduced in terms of an auxiliary variable $\hat{x}$ which records its update instants. Then, real time constraints on data, for instance periodicity or steadiness, are expressed by relating these dedicated variables and the current time. These auxiliary variables are also used to restrict the time shift between the source and the image of an observation: the semantics of the observation relation is extended to allow to relate the time behavior of a source and of an image by expressing different properties, such as the time lag between the current value of the image and its corresponding source value.

The real time constraints about data behavior can be specified by means of these timed observations as illustrated in an automotive speed control example.

Lastly, we discuss the possibility to check the consistency of a specification stated in terms of timed observations. A specification is consistent if and only if the verification process can construct correct executions.

*An earlier version of this paper was presented at the 3rd Real-Time Software Workshop, RTS2008, in Wisla, Poland, October 20, 2008.

†Université de Toulouse—IRIT, 2, rue Charles Camichel, 31071 Toulouse, FRANCE {tleberre, mauran, padiou, queinnec}@enseeiht.fr

However, the target systems are potentially infinite and an equivalent finite state transition system must be derived from the initial one before verification. The feasibility of this transformation is based upon assumptions about finite bounds of the time constraints.

2. State of the Art. We are interested in systems such as sensors networks. Our goal is to guarantee that the input data dispatched to processing units are timely sound despite the time shift introduced by the transit of data. Most approaches taken to check timed properties of distributed systems are based on studying the timed *behavior* of tasks. For example, works such as [10] propose to include the timed properties of communication in classical scheduling analysis.

Our approach is state-based and not event-based. We express the timed requirements as safety properties that must be satisfied in all states. The definition of these properties do not refer to the events of the system and is only based on the values of the system variables. We depart from scheduling analysis by focusing on the variables behavior and not considering the tasks and related system events. Our intent is to allow the developer to give a more declarative statement of the system properties, easier to write and less error-prone. Indeed, reasoning about state predicates is usually simpler than reasoning about a set of valid sequences of events.

Others approaches based on variables are mainly related to the field of databases. For example, the variables semantics and their timed validity domain are used in [12] to optimize transaction scheduling in databases. Our work stands at a higher level since we propose to give an abstract description of the system in terms of a specification of relations between data. For instance, our framework can be used to check the correctness of an algorithm with regards to the aging of the variables values. It can also be used to specify a system without knowing its implementation.

Similar works use temporal logic to specify the system. For example, in [2], OCL constraints are used to define the temporal validity domain of variables. A variation of TCTL is used to check the system synchronization and prevent a value from being used out of its validity domain. This work also defines timed constraints on the behavior and the relations between application variables, but these relations are defined using events such as message sending whereas our definitions are based on the variable values.

In [9], constraints between intervals during which state variables remain stable are defined by means of Allen's linear temporal logic. In other words, this approach also uses an abstraction of the data timelines in terms of stability intervals. However, the constraints remain logical and do not relate to real time. Nevertheless, the authors expect to apply this approach in the context of autonomous embedded systems.

Using a semantics based on state transition system, we give a framework which aims at describing the relations between the data in a system, and specifying the required timed properties of the system.

3. Theoretical settings.

3.1. State Transition System. Models used in this paper are based on state transition systems. Our work uses the TLA+ formalism [7], but this paper does not require any prior knowledge of TLA+. A *state* is an assignment of values to variables. A *transition relation* is a predicate on pairs of states. A *transition system* is a couple (set of states, transition relation). A *step* is a pair of states which satisfies the transition relation. An *execution* σ is any infinite sequence of states $\sigma_0\sigma_1 \ldots \sigma_i \ldots$ such that two consecutive states form a step. We note $\sigma_i \rightarrow \sigma_{i+1}$ the step between the two consecutive states σ_i and σ_{i+1}.

A *temporal predicate* is a predicate on executions; we note $\sigma \models P$ when the execution σ satisfies the predicate P. Such a predicate is generally written in linear temporal logic. A *state expression* e (in short, an expression) is a formula on variables; the value of e in a state σ_i is noted $e.\sigma_i$. The sequence of values taken by e during an execution σ is noted $e.\sigma$. A *state predicate* is a boolean-valued expression on states.

3.2. Introducing Time. We consider real time properties of the system data. To distinguish them from (logical) temporal properties, such properties are called *timed* properties. Time is integrated in our transition system in a simple way, as described in [1]: time is represented by a variable T taking values in an infinite totally ordered set, such as $\mathbb{N}$ or $\mathbb{R}^+$. T is an increasing and unbound variable. There is no condition on the density of time, and moreover, it makes no difference whether time is continuous or discrete (see discussion in [8]). However, as an execution is a sequence of states, the actual sequence of values taken by T during a given execution is necessarily discrete. This is the digital clock view of the real world. Note that we refer to the variable T to study time and that we do not use the usual timed traces notation.

An execution can be seen as a sequence of snapshots of the system, each taken at some instant of time. We require that there are "enough" snapshots, that is that no variable can have different values at the same time and so in the same snapshot. Any change in the system implies time passing.

Definition 3.1 (Separation) *An execution σ is separated if and only if for any variable x:*

$$\forall i, j : T.\sigma_i = T.\sigma_j \Rightarrow x.\sigma_i = x.\sigma_j$$

In the following, we consider only separated executions. This allows to timestamp changes of variables and ensures a consistent computation model.

3.3. Clocks. Let us consider a totally ordered set of values $\mathcal{D}$, such as $\mathbb{N}$ or $\mathbb{R}^+$. A clock is a (sub-)approximation of a sequence of $\mathcal{D}$ values. We note $[X \rightarrow Y]$ the set of all functions whose domain is X and whose range is any subset of Y.

Definition 3.2 (Clock) *A clock c is a function in $[\mathcal{D} \rightarrow \mathcal{D}]$ such that:*

- *it never outgrows its argument value:*
  $\forall t \in \mathcal{D} : c(t) \leq t$
- *it is monotonously increasing:*
  $\forall t, t' \in \mathcal{D} : t < t' \Rightarrow c(t) \leq c(t')$
- *It is lively:*
  $\forall t \in \mathcal{D} : \exists t' \in \mathcal{D} : c(t') > c(t)$

The predicate $clock(c)$ is true if the function c is a clock.

In the following, clocks are used to characterize the timed behavior of variables. They are defined on the values taken by the time variable T, to express a time delayed behavior, as well as on the indices of the sequence of states, to express a logical precedence.

4. Specification of Data Timed Behavior. We introduce here the relation and properties used in our framework to describe the properties that must be satisfied by a system. Our approach is state-based and gives the relation that must be satisfied in all states. We define the observation relation to describe the relation between variables. A way to describe the timed behavior of variables, that is properties of the history of data, is introduced. We then extend the observation relation to enable the expression of timed constraints on the behavior of system variables linked by observations. For that purpose we define predicates which bind and constraint relevant instants of the timeline of the source and the image of an observation. These predicates are expressed as bounds on the difference between two relevant instants.

4.1. The Observation Relation. We define an observation relation on state transition systems as in [5]. The observation relation is used to abstract a value correlation between variables. Namely, the observation relation states that the values taken by one variable are values previously taken by another variable or state expression.

In the basic case, the observation relation binds two variables, the source x and the image $`x$, and denotes that the history of the variable $`x$ is a sub-history of the variable x. The relation is defined by a couple $< source, image >$ and the existence of at least a clock that defines for each state which one of the previous values of the source is taken by the image. This definition is actually given to allow any state expression (a formula on variables) as the source[1]. The formal definition is:

Definition 4.1 (Observation) *The variable $`x$ is an observation of the state expression $e$ in execution $\sigma$: $\sigma \vDash `x \prec e$ iff:*

$$\exists\ c \in [\mathbb{N} \rightarrow \mathbb{N}] : clock(c) \wedge\ \forall i : `x.\sigma_i = e.\sigma_{c(i)}$$

[1] As we could introduce a new variable aliased to this expression, we often talk, in the following, of the source *variable*. This is to simplify the wording and the description.

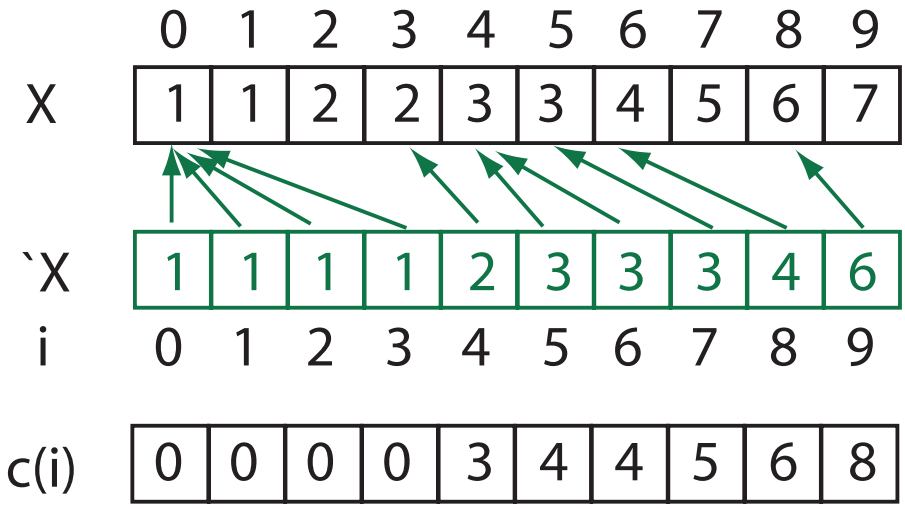


FIG. 4.1. *The Observation Relation*

This relation states that any value of $`x$ is a previous value of $e$. Due to the properties of the observation clock $c$, $`x$ is assigned e values in accordance with the chronological order. Moreover, c always eventually increases, so $`x$ is always eventually updated with a new value of $e$. Figure 4.1 shows an example of an observation relation binding two variables $x$ and $`x$.

The observation can be used to abstract communication in a distributed system, as well as to abstract computations:

- Communication consists in transferring the value of a local variable to a remote one. Communication time and lack of synchronization create a lag between the source and the image, which is modeled by $remote \prec local$.
- In state transition systems, an expression $f(X)$ models an instantaneous computation. By writing $y \prec f(X)$, we model the fact that a computation takes time and that the value of y is based on the value of X at the beginning of the computation. Here X can be a tuple of variables, according to the arity of f: given $X = \langle x_1, \ldots, x_n \rangle$, the observation $\sigma \models `x \prec f(X)$ means that $\exists\ c \in [\mathbb{N} \to \mathbb{N}] : clock(c) \wedge\ \forall i : `x.\sigma_i = f(x_1.\sigma_{c(i)}, \ldots, x_n.\sigma_{c(i)})$. As the same clock is used, all values of the inputs (X) are read at the same time, implying a synchronous behavior.
  Additional observation relations can be introduced to model an asynchronous reading of the inputs. For instance, $`a \prec a, `b \prec b, c \prec f(`a, `b)$ models a system where a and b are independently read (the first two observations), and then c is computed through a function f.

Note that the observation definition does not refer to real time and only models an arbitrary delay in terms of state sequences. Real time properties will now be introduced.

4.2. The Timeline of Variables. In order to state properties about the timed behavior of a variable x, we want to be able to refer to the last time x was updated. These are called the update instants and form its timeline $\hat{x}$. The definition of $\hat{x}$ is based on the history of the values taken by x and captures the instants when each value of x appeared, e.g. the beginning of each occurrence.

Definition 4.2 (timeline) *For a separated execution σ and a variable x, the variable $\hat{x}$ is the timeline of x and is defined by:*

$$\forall i : \hat{x}.\sigma_i = T.\sigma_{min\{j | \forall k \in [j..i] :\ x.\sigma_i = x.\sigma_k\}}$$

The timeline $\hat{x}$ is built from the history of x values and is a sequence of update instants. For a variable x and a state σ_i, the *update instant* of x in σ_i is defined as the value taken by the time T at the earliest state when the value $x.\sigma_i$ appeared and continuously remained unchanged until state σ_i.

Note that the developer may provide an explicit definition of $\hat{x}$, without having to describe the actual values of x, e.g. by stating that x is periodically updated.

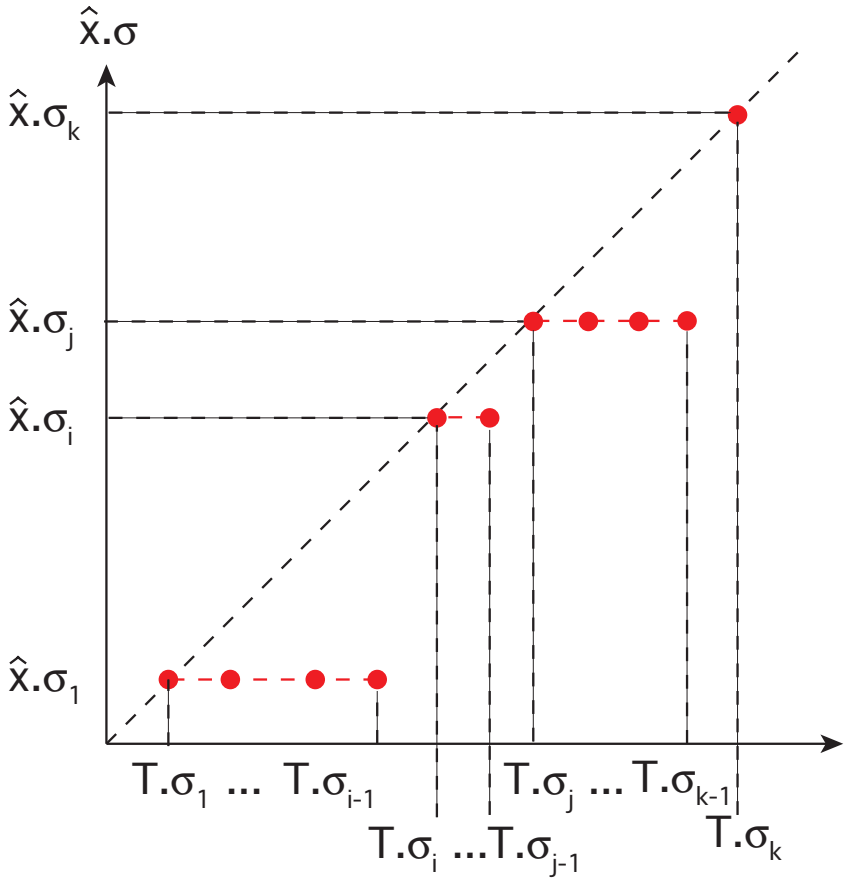


FIG. 4.2. *Graph of* $\hat{x}$

When x is updated and its value changes then the value of $\hat{x}$ is also updated. Conversely if $\hat{x}$ changes then x is updated. This property allows us to rely exclusively on the values of $\hat{x}$ to study the timed properties of x.

We also define the instant $Next(\hat{x})$ that returns, at each state, the next value of $\hat{x}$ and thus the next instant when the value of x is updated, i. e. the instant when the current value disappears. If x is stable at a state σ_i (no new update), then $Next(\hat{x}).\sigma_i = +\infty$.

As in the case of source variable versus source expression, the definition of a timeline $\hat{x}$, which is given for a variable x, is actually valid for a state expression. For the sake of clarity, we will once again talk of "variables" where "state expressions" could equally be used in the remainder of this section.

4.3. Behavior of Variables. The timeline $\hat{x}$ is used to describe the timed behavior of a variable x. In this paper, we focus on specific kinds of variables. We expect each value of each variable to remain unchanged for a bounded number of time units. We want to be able to express the minimum and the maximum duration between two consecutive updates. This allows to describe two basic behaviors: a sporadic variable keeps each value for a minimum duration, and on the contrary, a lively variable has to be updated often, no value can be kept longer than a given duration. These properties are formulated by bounds on the difference between $\hat{x}$ and $Next(\hat{x})$, using a property called *Steadiness* applied to a variable. These bounds denote how long each value of x can be kept.

Definition 4.3 (Steadiness) *The steadiness of a variable* x *in the range* $[\delta, \Delta]$ *is defined by:*

$$\sigma \vDash x \ \{Steadiness(\delta, \Delta)\} \triangleq \\ \forall i : \delta \leq Next(\hat{x}).\sigma_i - \hat{x}.\sigma_i < \Delta$$

$\Delta - \delta$ is the jitter on x updates. More elaborate properties can be derived from the steadiness property. For example, we can introduce a stronger property, periodicity, where no time drift is allowed.

Definition 4.4 (Periodicity) *A variable x is periodic of period P with jitter J and phase ϕ iff:*

$$\begin{array}{l}\sigma \vDash x\ \{Periodic(P,J,\Phi)\} \triangleq \\ \qquad\qquad x\ \{Steadiness(P-2J, P+2J)\} \wedge \\ \forall i : \exists n \in \mathbb{N} : \hat{x}.\sigma_i \in [\phi + nP - J, \phi + nP + J]\end{array}$$

Such a variable is updated around all instants $\phi + nP$. Note that J must verify $J < P/4$ to ensure that the variable is updated once and only once per period.

4.4. Timed Observation. We use the concept of timeline to extend the observation relation with timed characteristics. The timed constraints that extend the observation must capture the latency introduced by the observation and the timeline of the source to produce the timeline of the image. We define a set of predicates on the instants characterizing the source and the image timelines and the observation clock. Formally, a timed observation is defined as follows:

Definition 4.5 (Timed Observation) *A timed observation is defined as an observation satisfying a set of predicates.*

$$\begin{array}{l}\sigma \vDash {}`x \prec e \left\{ \begin{array}{l} Predicate_1(\delta_1, \Delta_1), \\ Predicate_2(\delta_2, \Delta_2), \\ \ldots \end{array} \right\} \triangleq \\ \qquad \exists c \in [\mathbb{N} \to \mathbb{N}] : clock(c) \wedge \\ \qquad\qquad\qquad \forall i : {}`x.\sigma_i = e.\sigma_{c(i)} \wedge \\ \qquad\qquad\qquad Predicate_1(c, \delta_1, \Delta_1) \wedge \\ \qquad\qquad\qquad Predicate_2(c, \delta_2, \Delta_2) \ldots\end{array}$$

The predicates that can be used to describe the timed properties of the relation between two variables are the following ones:

Definition 4.6 *Given a variable $`x$ and a state expression $e$ such that $\sigma \vDash {}`x \prec e$ with a clock $c \in [\mathbb{N} \to \mathbb{N}]$, the predicates are:*

$$\begin{array}{rcl} Lag(c,\delta,\Delta) & \triangleq & \delta \leq {}`\hat{x}.\sigma_i - \hat{e}.\sigma_{c(i)} < \Delta \\ Stability(c,\delta,\Delta) & \triangleq & \delta \leq Next(\hat{e}).\sigma_{c(i)} - \hat{e}.\sigma_{c(i)} < \Delta \\ Latency(c,\delta,\Delta) & \triangleq & \delta \leq T.\sigma_i - \hat{e}.\sigma_{c(i)} < \Delta \\ Medium(c,\delta,\Delta) & \triangleq & \delta \leq T.\sigma_i - T.\sigma_{c(i)} < \Delta \\ Freshness(c,\delta,\Delta) & \triangleq & \delta \leq T.\sigma_{c(i)} - \hat{e}.\sigma_{c(i)} < \Delta \\ Fitness(c,\delta,\Delta) & \triangleq & \delta \leq Next(\hat{e}).\sigma_{c(i)} - T.\sigma_{c(i)} < \Delta \end{array}$$

When no lower (resp. upper) bound is significant, 0 (resp. $+\infty$) should be used.

These predicates have to be true at every state and every instant. The definition of an observation is done by stating which predicates must be satisfied. So far, this set has been sufficient to express the different behaviors that we had to analyze, but it can be extended.

- Predicate *Lag* is used to bound the duration between an update of the source and an update of the image. An upper bound states that, when the image is updated, it must be updated with an expression of source that was updated in a recent time. A lower bound states that when there is an update of the source, the new value cannot be used to update the image before the lower bound has elapsed.
- Predicate *Latency* bound in each state the time elapsed since the assignment of the image's current value on the source.
- Predicate *Stability* is used to filter sources values depending on their duration. For example we can eliminate transient values and keep sporadic ones, or the contrary.
- The observation clock and the difference $i - c(i)$ give the logical delay introduced by the observation. Predicate *Medium* bounds the temporal delay related to this logical delay. So the bounds state that there must exist a logical delay inducing a temporal delay satisfying the bounds, i. e. in each state, there must be one previous state so that the time elapsed since that state is below this upper bound and above the lower bound and so that the image's current value was assigned on the source. A lower bound can be used to state that a value of the source cannot appear on the source before this lower bound has elapsed and so this bounds denotes a communication or computation time.

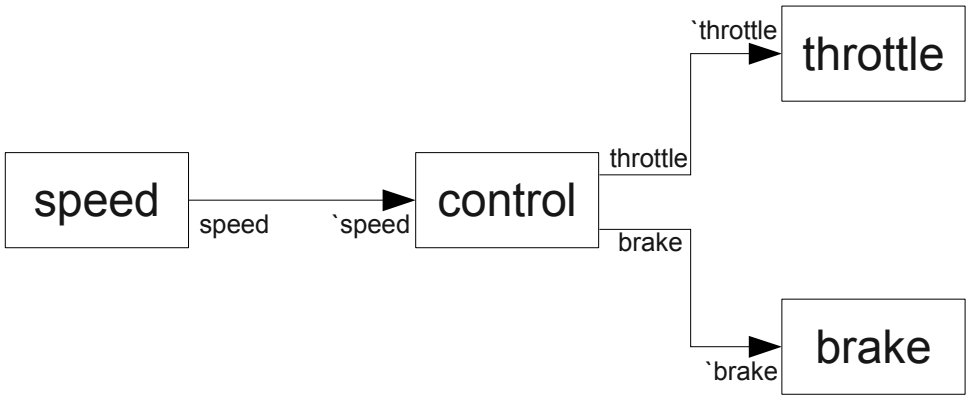


FIG. 5.1. *Cruise Control System*

- Predicates *Freshness* and *Fitness* are used to define intervals of time, relative to the update instants, that the observation clock is prevented to refer. So the logical delay that satisfies the predicate Medium must refer to an instant that satisfies the Freshness and Fitness predicates. An upper bound on Freshness prevents states where the value of the source is not fresh anymore to be referred. For example, a conjunction of Medium and Freshness predicates states that the current value of the image must have been available on the source recently and that it was still fresh at these instants. On the contrary, a lower bound on Freshness denotes an impossibility to access a value just after its assignment. Fitness allows or forbids the states depending on the time remaining until the source value is updated. A lower bound prevents to refer to a state where the value is about to be updated.

Note that, at the beginning of an execution, some predicates such as *Medium* cannot be satisfied. In order to address this problem, the timed predicates do not have to be satisfied in initial states. The image values are replaced by a given default value. This extension is similar to the "followed by" operator $\rightarrow$ in Lustre [6].

5. Specifying a System in Terms of Timed Observations.

5.1. A Brief Description. As an example, we consider a simplified car cruise control system. The goal of such a system is to control the throttle and the brakes in order to reach and keep a given target speed. The system is composed of several interacting components (see Figure 5.1):

- a speed monitor, which computes the current speed, based on a sensor counting wheel turns;
- the throttle actuator, which controls the engine;
- the brakes, which slow down the car;
- the control system which handles the speed depending on the current and the chosen speed;
- a communication bus which links the devices and the control system.

The environment, the driver, and the engine influence the speed of the car. Once the cruise control is activated and a target speed is chosen, the control system can choose either to accelerate by increasing the voltage of the throttle actuator or to decelerate by decreasing this voltage and by using brakes. In order to ensure a reactive behavior, each command issued by the cruise control system must be carried out within a given time limit.

Each component uses and/or produces data. We use observations to specify the system and characterize correct executions.

5.2. Data and Observations. Firstly, we define the state variables of the cruise control system, and we bind these variables using observation relations.

The speed monitor computes the values of a variable *speed*, and these values are sent to the control system as a variable $`speed$. We express this as an observation $`speed \prec speed$.

The choices of the control system are based on the current speed and more precisely on the value of $`speed$. Two functions are used to compute the values used as inputs by the brakes and by the throttle actuator. Using the speed values, we compute the values of two variables: $throttle \prec control1(`speed)$ and $brake \prec control2(`speed)$.

Lastly, the values of *throttle* and *brake* are delivered to dedicated devices into variables $`throttle$ and $`brake$, such that $`throttle \prec throttle$ and $`brake \prec brake$.

- variables behaviors:

$$speed\ \{Steadiness(\delta_1, +\infty)\}$$
$$throttle\ \{Steadiness(\delta_2, +\infty)\}$$
$$brake\ \{Steadiness(\delta_3, +\infty)\}$$

- communications:

$$`speed \prec speed\ \{Medium(\delta_4, +\infty)\}$$
$$`throttle \prec throttle\ \{Medium(\delta_4, +\infty)\}$$
$$`brake \prec brake\ \{Medium(\delta_4, +\infty)\}$$

- computations:

$$throttle \prec control1(`speed)\ \{Medium(\delta_5, +\infty)\}$$
$$brake \prec control2(`speed)\ \{Medium(\delta_6, +\infty)\}$$

- complete processing chains:

$$`throttle \prec control1(speed)\ \{Latency(0, \Delta)\}$$
$$`brake \prec control2(speed)\ \{Latency(0, \Delta)\}$$

FIG. 5.2. *System Specification*

5.3. Requirements and Properties. We express the requirements and known timed properties of the system, and we state them as characteristics of the system variables and observations. These characteristics are given in Figure 5.2.

The speed is computed using the ratio of the number of wheel turns to the elapsed time. A minimum time is required to produce a significant result. Thus, there must be a minimum time δ_1 between each update of *speed*. Also, due to scheduling constraints, there must be a minimum time δ_2 (respectively δ_3) between each computation and update, of *throttle* (respectively *brake*).

Each communication on the bus takes a minimum transit time, regardless of the communicating protocol that is chosen. Predicate *Medium* (see Definition 4.6) is used to define a lower bound on the observations expressing communication. Similarly, we represent the minimum computation time of functions *control*1 and *control*2, by means of predicate *Medium*.

We expect each data to be used soon enough after each update. More precisely, we want each command issued to the brake or to the throttle to be based on fresh values of the speed. Thus, we require the complete processing chain to be completed in a short enough time.

A composition of observations is an observation, for example if $y \prec x$ and $z \prec f(y)$ then $z \prec f(x)$ [5]. We use this property to define the processing chains relating *`throttle* and *`brake* to *speed*, via *'speed* as observations, which enables us to express the requirements on the duration of the processing chains as upper bounds of *Latency* predicates (see Definition 4.6) on these observations. Note that, although the *Latency* upper bound (Δ) is the only upper bound given in the system specification, it implicitly sets upper bounds on the *Medium* and *Steadiness* characteristics of the other observations and variables of valid executions.

5.4. Case Study Analysis. The goal of the analysis is to prove that the specification is consistent and that there is at least one execution satisfying the requirements. In our example, a nonempty set of valid executions ensures the availability of timely sound values. From this set, we can deduce the required update frequency of the *speed* variable. For example, we check the existence of a maximum time acceptable between each update. We analyze the admissible values of the *Medium* to deduce the communication and computation times that are permitted. Then, we determine the possible values of the observation clocks in the states corresponding to the timeline of the image. These values give the instants at which the values of the source are caught and so, for example the instants when a message must be sent or when a computation must start.

For all these properties, a choice must be done. For example, choosing a set of executions may alleviate the bounds on communication time but then reduce the instants when the message must be sent.

6. System Analysis. We give here properties of our framework based on observations in order to carry out an analysis. A system specified with observation relations must be analyzed to check the consistency of the specification, i. e. if there exists an execution satisfying the specification.

We discuss the analysis method in a discrete context. The semantics of the specification is restricted by

discretizing time: i. e. the values taken by time T are in $\mathbb{N}$. For discussion about the loss of information using discrete time instead of dense time and defending our choice, see [8] for example.

6.1. Feasibility of a Specification. Given a specification based on our framework, the value of T is unbounded and we have no restriction on the values that can be taken by variables. Therefore the system defined by the specification is infinite. Nevertheless, we can build a finite system equivalent to the specification for the timed properties studied with this framework. This allows us to model-check the consistency of the specification in a finite time. Here are the main principles of this proof. The definition of a finite system bisimilar to the original one is based on two equivalence relations.

Since the scope of this framework is to check the satisfaction of timed requirements, we focus on the auxiliary variables used to describe the timeline of each application variable. We define a system where only variables denoting instants are kept, i. e. the variable describing the timelines and the observation clocks. The states and transitions of the system are defined by the values of these variables and the satisfaction of observations and variables properties. Allowed states and transitions do not depend on the values that can be taken by each variable but on the instants describing their timeline and on the observation clocks. Thus, when we build a system where only these instants are considered, we do not lose or add any characteristics about the timed behavior of the system. We define an equivalence where two states are equivalent if and only if the observation clocks and the timeline variables are equal. This equivalence is used to build a bisimilarity relation between the specified system and the one built upon only the instants.

The second reason preventing to consider a bounded number of states is the lack of bound on time. The values of the timelines and observation clocks are also unbounded. In order to reduce the possible values that can be taken by the system variables denoting instants, we define a system where all values of the instants are stored modulo the length of an analysis interval. We denote this number as L. L must be carefully chosen, greater than the upper bounds on the variables *Steadiness* and the observations *Latency* characteristics and it has to be a multiple of the variable periods.

Such a number L only exists if all variables and observations have upper bounded characteristics. When the source of an observation is bounded and so is the observation, such a bound is deduced for the image. Restricting the behavior by expecting variables to be frequently updated and the shift introduced by distribution to be bounded seems consistent for such real time systems.

In the system defined by the specification, transitions are based on differences between the instants characterizing the variable timelines. These differences cannot exceed the chosen length L. Thus, for each state, if the value of the time T is known and if the values of the other variables are known modulo L, then for each variable there is only one possible real value that can be computed using the value of T. Consequently, considering the clock values modulo this length does not add or remove any behavior of the original system. We define an equivalence where two states are equivalent if the timelines and the observation clocks are equal modulo L. A system built by considering all values modulo L is bisimilar with the original system using this equivalence.

Based on these two equivalences, we build a system by removing variables which do not denote timelines or observation clocks and by considering the values modulo L. This system is bisimilar to the specification and preserves the timed properties. Since all values are bounded by the length of the analysis interval and there is a bounded number of values, it defines a system with a bounded number of states. This result proves the decidability of the framework for the verification of safety properties that can be done using the finite system.

6.2. Complexity. We have proved the existence of a finite system equivalent to our system. We give here the complexity of a process to effectively build this equivalent finite system. In order to build a transition from a state to a new state, we build a set of inequalities deduced from the properties of the previous state and from the observations and variables properties. To solve this set of inequalities and deduce the possible values of instant variables in the new state, we use difference bound matrices [4]. Considering a system where n variables are studied, the size of each matrix is $O(n^2)$, and the complexity for reducing it to its canonical form and building the new state is $O(n^3)$ [4]. The maximum number of states to build depends on all possible combinations of values taken by variables. Each timed variable can take values between 0 and L and the number of instant variables is a multiple of n, so we have $O(L^n)$ states. Lastly, the complexity to build the system is $O(n^3 * L^n)$. Considering the memory, we have to store $O(L^n)$ states and $O(L^{2n})$ transitions. Therefore this direct approach is technically feasible only with small enough systems. The complexity is more heavily impacted by the number of variables (n) than by the analysis interval (L).

6.3. Verification of an Implementation. A second goal is to check that an implementation is correct with regard to a specification based on observations. This approach is fully described in [13] and is only hinted here.

As all timed properties are safety properties, an implementation is correct if no execution deadlocks (so as to ensure liveness) and all its executions are included in the executions defined by the specification.

In order to check the satisfaction of the specification by an implementation, we give a model of the specification in the same semantics we use to model an implementation. Such a model is described by defining elementary transitions. An elementary transition relation models the evolution of the values states of availability in the observation relations of the system. These elementary transition relations are used to build the variable transition relation of the image of an observation. The variable transition relations are then used to build the global transition relation.

Once both the specification and the implementations have been translated into such transition relations, we must verify that the model of the specification simulates the implementation. In order to check this property, we build a state transition system similar to the synchronized product of labelled transition systems. The actions are used as labels on the transitions of the systems.

6.4. Other Approaches. Since our approach relies on the TLA+ formalism, we could have used the dedicated tool TLC, the TLA+ model checker. A logical definition of the observation requires the temporal existential quantifier $\exists$, which is not implemented in TLC. Therefore a concrete definition of the observation based on an explicit observation clock has been used. It is only after we have reduced the system to a finite one that a model checker such as TLC could be used.

To be able to more precisely characterize executions satisfying the specification, we currently explore methods to build these executions more easily. A first proposal is to reduce the complexity of such a process by relying on proofs on system properties. The proof approach can easily be used only under certain conditions and in order to proceed to some system simplifications. For example, a periodic source induces properties for its image through an observation. Using these properties reduces the number of states we have to build by forecasting some impossible cases. Proving the full correctness of the system is possible but it is complex and it has not been automatized yet.

Another way is to use controller synthesis methods [3]. Properties of the observation can be expressed as safety properties using LTL and be derived as Büchi automata [11]. Two automata describe the behavior of the source and the image of an observation, exchanging values through a queue. Restrictions can be added to introduce the used implementation and its compatibility with executions defined by the specification. The complexity of controller synthesis methods has still to be explored.

7. Conclusion. We propose an approach focused on variables instead of tasks and processes, to model and analyze distributed real time systems. We specify an abstract model postponing task and communication scheduling. Based on the state transition system semantics extended by a timed referential, we express relations between variables and the timed properties of variables and communications. These properties are used to check the freshness of values, their stability, and the consistency of requirements. A possible analysis is to build a finite system bisimilar to the specified one. The results are used to help implementation choices.

Perspectives are to search other methods that decrease the complexity of the analysis of a specification and to use this approach with different examples to expand the number of available properties and increase expressiveness. We also work on using analysis results to help generating an implementation satisfying the specification.

REFERENCES

[1] M. Abadi and L. Lamport, *An old-fashioned recipe for real time*, ACM Transactions on Programming Languages and Systems, 16 (1994), pp. 1543–1571.

[2] S. Anderson and J. K. Filipe, *Guaranteeing temporal validity with a real-time logic of knowledge*, in ICDCSW '03: Proc. of the 23rd Int'l Conf. on Distributed Computing Systems, IEEE Computer Society, 2003, pp. 178–183.

[3] E. Asarin, O. Maler, and A. Pnueli, *Symbolic controller synthesis for discrete and timed systems*, in Hybrid Systems II, London, UK, 1995, Springer-Verlag, pp. 1–20.

[4] J. Bengtsson and W. Yi, *Timed automata: Semantics, algorithms and tools*, in Lecture Notes on Concurrency and Petri Nets, W. Reisig and G. Rozenberg, eds., Lecture Notes in Computer Science vol 3098, Springer–Verlag, 2004.

[5] M. Charpentier, M. Filali, P. Mauran, G. Padiou, and P. QuÃľinnec, *The observation : an abstract communication mechanism*, Parallel Processing Letters, 9 (1999), pp. 437–450.

[6] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, *The synchronous data-flow programming language LUSTRE*, Proceedings of the IEEE, 79 (1991), pp. 1305–1320.

[7] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, 2002.

[8] E. Lee and A. Sangiovanni-Vincentelli, *A framework for comparing models of computation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17 (1998), pp. 1217–1229.

[9] G. Roşu and S. Bensalem, *Allen Linear (Interval) Temporal Logic—Translation to LTL and Monitor Synthesis*, in International Conference on Computer-Aided Verification (CAV'06), no. 4144 in Lecture Notes in Computer Science, Springer Verlag, 2006, pp. 263–277.

[10] K. Tindell and J. Clark, *Holistic schedulability analysis for distributed hard real-time systems*, Microprocessing and Microprogramming—Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems), 40 (1994), pp. 117–134.

[11] M. Y. Vardi, *An automata-theoretic approach to linear temporal logic*, in Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 238–266.

[12] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley, *Scheduling transactions with temporal constraints: exploiting data semantics*, in RTSS '96: Proc. of the 17th IEEE Real-Time Systems Symposium, 1996, pp. 240–253.

[13] Tanguy Le Berre, Philippe Mauran, Gérard Padiou, and Philippe Quéinnec, *A data oriented approach for real-time systems*, in IEEE Int'l Conf. on Real-Time and Network Systems RTNS09, 2009.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 241–252. http://www.scpe.org
ISSN 1895-1767

STUDY OF DIFFERENT LOAD DEPENDENCIES AMONG SHARED REDUNDANT SYSTEMS

JÀN GALDUN*, JEAN-MARC THIRIET*, AND JÀN LIGUŠ†

Abstract. The paper presents features and implementation of a shared redundant approach to increase the reliability of networked control systems. Common approaches based on redundant components in control system use passive or active redundancy. We deal with quasi-redundant subsystems (shared redundancy) whereas basic features are introduced in the paper. This type of redundancy offers several important advantages such as minimizing the number of components as well as increasing the reliability. The example of a four-rotor mini-helicopter is presented in order to show reliability improving without using any additional redundant components. The main aim of this paper is to show the influence of the load increasing following different scenarios. The results could help to determine the applications where quasi-redundant subsystems are a good solution to remain in a significant reliability level even if critical failure appears.

Key words: shared redundancy, dependability, networked control systems

1. Introduction. To be able to obtain relevant results of reliability evaluations for complex systems, it is necessary to describe the maximum of specific dependencies within the studied system and their influences on the system reliability. Different methods or approaches for control systems' reliability improvement are developed in order to be applied to specific subsystems or to deal with dependencies among subsystems. A classical technique consists in designing a fault-tolerant control [1] where the main aim is to propose a robust control algorithm. Guenab and others in [2] deal with this approach and reconfiguration strategy in complex systems, too.

On the other side is the design of reliable control architectures. Probably the most used technique is to consider the redundant components which enlarge the system structure and its complexity too. Active and passive redundancy is the simplest way how to improve dependability attributes of the systems such as reliability, maintainability, availability, etc [3]. However, as it was mentioned the control structure turns to be more complex due to an increasing number of components as well as the number of possible dependencies among components, it is in particular the case for Networked Control Systems [4] [5].

The paper introduces complex networked control architecture based on cascade control structure. The cascade structure was chosen purposely due to its advantages. This structure is widely used in industrial applications thanks to positive results for quality of control which are already described and generally known [6]. On the other side it offers some possibilities of system reliability improvement. There are potentially redundant components such as controllers (primary, secondary). If more than one network is implemented we could consider them as potentially redundant subsystems too. Finally if the physical system allows it, it is possible to take profit from sensors. The cascade structure and other features are introduced in more details in the third part.

The paper is organised as follows. After bringing closer the research background, the shared redundancy is introduced. The controllers and networks are presented in more details in order to show some dependencies which could be appeared when a shared redundancy approach is implemented. In the next part are presented networked topologies considered as cascade control (CC) structure of the 4-rotor mini-helicopter (drone) model [7]. Using Petri nets were prepared the models of the introduced quasi-redundant components as well as drone's control structure. A simple model of the two quasi-redundant subsystems is evaluated. Finally, are proposed the simulation results of the mentioned simple two components model as well as the model of the complex drone's structure with short conclusion.

2. Research Background. Control architecture design approach was taken into account by Wysocki, Debouk and Nouri [8]. They present shared redundancy as parts of systems (subsystems) which could replace another subsystem in case of its failure. This feature is conditioned with the same or similar function of the subsystem. Wysocki et al. introduce the shared redundant architecture in four different examples illustrated on "X-by-Wire" systems used in automotive applications. Presented results shown advantages of this approach in control architecture design.

The shared redundancy approach involves the problematic of a Load Sharing [9]. Thus, some of the components take part of the load of the failed components in order to let the system in functional mode.

*Laboratoire GIPSA-Lab (GIPSA-Lab UMR 5216 CNRS-INPG-UJF) BP 46, F-38402 Saint Martin d'Hères Cedex, France
†Department of Cybernetics and Artificial Intelligence, Technical University of Košice, Letnà 9, 04012 Košice, Slovakia

Consideration of the load sharing in mechanical components is presented by Pozsgai and others in [10]. Pozsgai and others analyze this type of systems and offer mathematical formalism for simple system 1-out-of-2 and 1-out-of-3. Also there are some mathematical studies [9] of several phenomena appeared on this field of research. Bebbington and others in [9] analyze several parameters of systems such as survival probability of load shared subsystems.

3. Shared Redundancy. Specific kind of redundant subsystems which have similar features such as active redundancy however gives us some additional advantages which will be introduced in further text. This kind of spares represents another type of redundant components which are not primary determined as redundant but they are able to replace some other subsystems if it is urgently required. This type of redundancy is referred as *shared redundancy* [8] or *quasi-redundancy* [11]. Due to its important advantages it is useful to describe this kind of spares in order to show several non-considered and non-evaluated dependencies which could have an influence to the system reliability. Identification and description of this influence should not be ignored in order to obtain relevant results of the reliability estimation of the systems which involve this kind of spares.

As it was mentioned above, the *shared redundancy* (SR) mentioned by Wysocki and others in [8] is in further text taken into account in the same meaning as a *quasi-redundant* (QR) component. Thus, quasi-redundant components are the parts of the system which follow their primary mission when the entire system is in functional state. However, when some parts of the system fail then this function could be replaced by another part which follows the same or a similar mission, thus by quasi-redundant part. The quasi-redundant components are not primary determined as active redundant subsystem because each one has its own mission which must be accomplished. Only in case of failure it could be used. In NCS appears the question of logical reconfiguration of the system when the data flow must be changed in order to replace the functionality of a subsystem by another one. For example, some new nodes will lose the network connection and the system has to avoid the state when packets are sent to a node which does not exist. Thus, the main features of the shared redundancy could be summarized as follows:
"Quasi-redundant component is not considered as primary redundant component such as the active or the passive redundant components."

Generally in networked control systems, three kinds of quasi-redundant components (subsystems) could be considered:

- QR controllers.
- QR networks.
- QR sensors.

Hence, a necessary but not sufficient condition is that a control structure where SR could be considered has to be composed at least of two abovementioned subsystems (controllers, networks, actuators). The subsystems should have similar functionality or construction in order to be able to replace the mission of another component. In case of quasi-redundant components there are several limitations. In order to take profit of quasi-redundant networks, it is necessary to connect all nodes in all considered QR networks. Thus, in case of different networks the components should have implemented all necessary communication interfaces. In case of QR controllers the hardware performance has to allow implementing more than one control task.

Third mentioned components are sensors. Consideration of the sensors as QR components has important physical limitations. In order to be able to replace a sensor for measuring a physical value X by another one for measuring Y it is necessary to use "multi-functional" smart sensors. *We can suppose that some combination of the physical values can not be measured by using one sensor due to the inability to implement the required functionality in one hardware component.*

Other limitation is the distance between failed sensor and its QR sensor which could have a significant influence to the possibility of its replacing. Generally, implementation of the QR sensors within control system structure could be more difficult than the application of the SR approach on controllers or networks.

There are several naturally suitable control structures which could implement the shared redundancy approach without other modifications such as cascade control structure (Fig. 3.1). This structure is often used in industrial applications thanks to its important features which improve the quality of control. With using cascade a control structure there are several constraints [8]. The main condition requires that the controlled system must contain a subsystem (secondary subsystem FS(s)—Fig. 3.1) that directly affect to the primary system FP(s). Thus, the cascade structure composes of two independent controllers which can be used in order to implement the shared redundant approach.

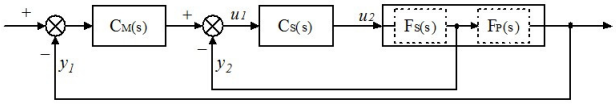


FIG. 3.1. *Main structure of the cascade control*

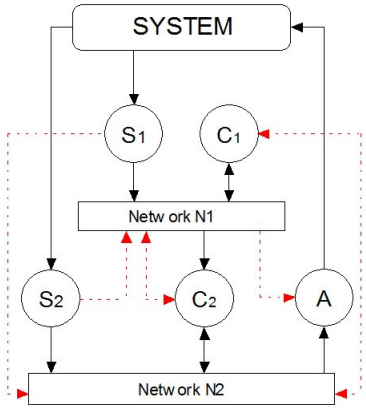


FIG. 3.2. *NCCS with two networks and alternative network connections*

Usually for secondary subsystems there is a condition of faster dynamics than primary process. This condition must not be fulfilled [8]; in this case, some modifications of conventional cascade structure (Fig. 3.1) and control laws must be provided.

3.1. Quasi-redundant controllers. In the previous text, several suitable control structures were briefly introduced. As it was shown the controllers covered by these structures could be considered as quasi-redundant components by default. Thus, the hardware of both components could be shared in order to implement a shared redundant approach.

Let's consider the networked cascade control system shown in figure 3.2. The system is composed of five main components (Sensor S_1, S_2, controllers C_1, C_2 and actuator A) and two networks. The communication flow among components is determined by its cascade control structure. Thus, sensor S_1 sends a measured value to controller C_1 (Master), the controller C_2 (Slave) receives the values from the sensor S_2 as well as the controller C_1 in order to compute an actuating value for the actuator A.

Each part of the system (components and networks) presents independent subsystem. However, when quasi-redundant components are studied, the system is not considered as composed of independent components. Depending on the performance parameters of the used hardware equipment in the control loop, a specific influence on the system reliability should be taken into account. Thus some dependencies should not be ignored in the dependability analysis. In the NCCS shown in Fig. 3.2 we could consider controllers C_1 and C_2 as the quasi redundant subsystems (components). Both QR controllers have a primary mission which should be followed. Thus, a controller C_1 controls outer control loop and controller C_2 stabilizes inner control loop. However in case of failure of one of them, we could consider the second one as a kind of spare.

As it was mentioned previously, the controllers follow their primary mission stabilization or performance optimization of the controlled system. Therefore, in regards to the similar hardware, it allows sharing the computing capacity and executing different tasks. Thus, in order to implement the SR approach, both controllers have to encapsulate both control tasks—for the outer and the inner control loop (see the cascade control structure in figure 3.1).

In non-failure mode the primary task is executed in both controllers. However, in case of controller's failure (primary or secondary) non-failed controller starts execute both tasks and computes actuating value for primary

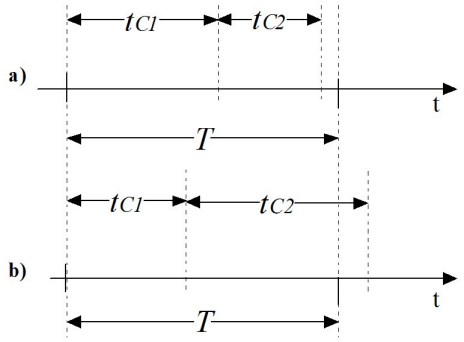


FIG. 3.3. *Possible scenarios for quasi-redundant controllers*

as well as secondary subsystems. In this case we can suppose two scenarios.

The first one supposes that the controller is able to execute all the necessary tasks within the required sample periods (Fig. 3.3a). Thus, no delays or other undesirable consequences are expected. In this case the behavior of the quasi-redundant component is similar as in the case of active redundant components. Thus, in the case of failure of one of the components, the second takes care about its mission until its failure.

Figure 3.3b shows a second case when time to execute both necessary tasks is greater than the required sampling period. Thus, the controller will cause the delays which have significant influence to the system stability [12] [13]. Therefore, this delay could be known which allows its partially compensating by using several methods [14]. Thus, we can suppose that the system destabilization will not occur immediately after the first delay and we are able to compensate it for some time interval. Thus, quasi-redundant controller does not fail immediately but its reliability decreased.

There are several situations when this scenario could be considered. In critical systems where the failure of an important component could cause undesired damages or other dangerous consequences, the shared redundancy approach could help to allocate some time interval in order to maintain the system in a safe state. Thus, the SR approach can be a significant technique to secure the system before a damage risk.

3.2. Quasi-redundant networks. The second part of the NCS which could be taken into account as SR subsystems are networks. Let's suppose a system with two networks (Fig. 3.2) where all components could communicate (connect) on these networks (N_1 and N_2) if it is needed. In this case we can apply the SR approach on this system.

Considered functionality of the quasi redundant networks is as follows. Both networks transmit required data—network N_1 transmit data from S_1 to C_1 and from C_1 to C_2 such as network N_2 from S_2 to C_2 and from C_2 to A. Thus both networks are active and allocated during the system mission. The same as in the case of QR controllers: when a network failed, the second one can take its load after a system reconfiguration. Thus, all required data are sent through the second network. Hence, two similar scenarios as with the controller task execution could be described. The amount of transmitted data on the network with a specified bit rate has logically influence on the probability of failure of the network (of course this depends on the network type and other parameters mentioned). This influence could be ignored when the network performance parameters are sufficient. However, we can suppose that the probability of network failure is increasing simultaneously when the network load increases.

The characteristic between network loading and its bit rate depends on the network type and have to be measured in real network conditions in order to determine the type of dependency—linear or nonlinear.

Not only the network bit rate can be important however other network limitations such as maximal number of nodes connected to the network, etc. All limits of the QR subsystems can create dependencies with direct influence on the system reliability. Primary, we could consider these dependencies as undesirable but in case of critical failures this SR approach gives some time to save the system.

When NCS with an SR approach are analyzed, this characteristic should be included in the prepared model and further evaluated in order to determine its influence to the reliability of the whole NCS.

3.3. Different scenarios in shared redundancy. When certain dependencies are ignored we could regard on the control system with QR components as a control structure with active redundant components.

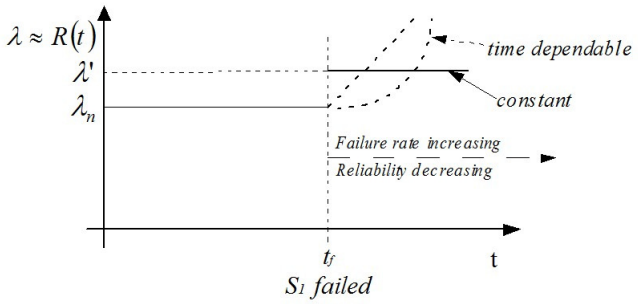


FIG. 3.4. *Possible failure rate curves for the subsystem S_2 during its mission*

However, there are several important scenarios when the reliability of the system could be decreased in order to prevent dangerous consequences or other undesirable events.

These scenarios could appear when some conditions could not be fulfilled (insufficient execution time or network bit rate) but the system need some time in order to take a safe state. Hence, it is necessary to identify and describe the influence of these dependencies which leads to more relevant results. Thus, prevent from too pessimistic or too optimistic results of the reliability analysis of the considered systems. The dependencies could be distinguished as follows:

- active redundant dependency,
- single step change of the nominal failure rate $\lambda_n \in \langle 0; 1\rangle$ increased once by a constant value—step load change,
- time depend change of the nominal failure rate λ_n -functional dependency- the load of the subsystem is changed with time passed from speared subsystem failure,
  1. linear,
  2. nonlinear.

Let's assume that the destabilization of the system does not occur immediately after the first delay on the network caused by insufficient controller's hardware or network's parameters. Thus, the quasi-redundant controller does not fail immediately but in this case its failure rate increases which correspond consequently to a decreased reliability.

Thus, in case of the active redundant dependency we suppose that a quasi-redundant subsystem has sufficient capacities in order to follow its primary mission as well as the mission of the failed subsystem (or subsystems).

A single step change of the nominal failure rate of the subsystem is considered in the case of subsystems where the failure rate of the quasi-redundant subsystem is changed (increased) once by a constant value (Fig. 3.4) during its life time. Thus, the new increased failure rate $\lambda^{'}$ remains constant during further life time of the subsystem. For example, let's suppose a NCS with two Ethernet networks where one of them has failed and consequently the system is reconfigured and all nodes (components) start to communicate through the non-failed network which has a sufficient bit rate capacity in order to transmit all the required data. However, the amount of data has been increased which consequently increases the probability of packets' collisions (under the assumption of a classical CSMA/CD protocol, for instance). Thus, the probability of failure (failure rate) has been increased up to the new value $\lambda^{'}$.

A third case considers the change of the nominal failure rate λ_n which depends on the time passed from the moment of the failure until current time of the working of the quasi-redundant subsystem which encapsulates the executing necessary tasks (own tasks as well as tasks of the failed subsystem). Thus, a functional dependency has to be considered. This dependency of the change of the failure rate λ_n could be described by a linear or nonlinear dependency / function. We could study the previous example of the system with two networks. However, in this case the bit rate of the second (non-failed) network is not sufficient. Consequently delays in data transmission as well as other consequential undesirable problems such as system destabilization might be caused. We can suppose that the non-failed network will fail in some time. Thus, the nominal failure rate λ_n of the second network is now time dependent and is linearly or nonlinearly increased until the system failure. Mentioned examples with related equations are further discussed in more details.

Let's suppose that the reliability of the system $R(t)$, probability of the failure during time interval $\langle 0; t\rangle$, is characterized by a nominal failure rate $\lambda_n \in \langle 0; 1\rangle$. Let's suppose a system with two subsystems S_1 and S_2

(such as the networks in the previous examples) whereas the subsystem S_1 will fail at first and then the quasi-redundant subsystem S_2 will follow both missions (S_1 and S_2). In figure 3.4 are shown two above mentioned scenarios when the nominal failure rate λ_n of the subsystem is increased by a constant value or by a value which could be described as a linear or nonlinear function (functional dependencies).

At first increasing the failure rate λ_n one time by a constant value (see Fig. 3.4) will be dealt. It corresponds to the reliability reduction of the quasi-redundant subsystem S_2 by increasing the failure rate, during its mission, from its nominal value λ_n up to new $\lambda^{'}$. Consequently, the system will follow its primary mission thanks to the QR subsystem S_2 but its failure rate is already increased and consequently the probability of failure of S_2 is higher. The difference between nominal λ_n and increased $\lambda^{'}$ failure rate will be called decrease factor d_R. Thus, the mentioned constant value is characterized by the decrease factor d_R of the QR subsystem and a new changed failure rate $\lambda^{'}$ at the fail time t_f is given by the followed simple formula:

$$\lambda^{'} = \lambda_n + d_R \tag{3.1}$$

The failure rate increases only one time by the specified value and the QR subsystem S_2 with a new constant failure rate $\lambda^{'}$ will follow both missions of its own mission and mission of the failed subsystem S_1.

The second case shown in figure 3.3 considers the reliability reduction where the failure rate λ_n is increased during the working of the subsystem S_2 by a specified decrease factor. This change of the nominal failure rate depends on time whereas with time extending the failure rate of the S_2 is got near to 1 (system failed). Thus, a decrease function $f_{d_R}(t)$ is represented by a linear or nonlinear characteristic and depends on the real subsystem which is considered as quasi-redundant. Thus, an increased failure rate $\lambda^{'}$ of the subsystem S_2 depends on time t and is given by the following formula:

$$\lambda^{'}(t) = \lambda_n + f_{d_R}(t) \tag{3.2}$$

As it was mentioned, the decrease function $f_{d_R}(t)$ can be represented by a simple linear function, for example,

$$\lambda^{'}(t) = \lambda_n + d_R 10^{-3}(t + 1 - t_f) \tag{3.3}$$

where $t + 1$ allows changing the nominal failure rate λ_n at the moment of the failure at time t_f.

On the other side a nonlinear exponential function can be considered as follows:

$$\lambda^{'}(t) = \lambda_n + e^{d_R(t-t_f)} \tag{3.4}$$

where $\lambda^{'}$ is the value of the increased failure rate, λ_n is the nominal failure rate of the component, t_f is the time of the failure of the component, d_R is the decrease factor which has a direct influence on the increased failure rate.

3.4. Application to a mini-drone helicopter. The NCC structure is applied for the control of a four rotors mini-helicopter (Drone, Fig. 3.5). The proposed control structure for this real model is as follows. The NCC architecture is composed of one primary controller (Master) and one secondary controller (Slave), thirteen sensors, four actuators and two communication networks.

The Master is designed for attitude stabilization (control) through Slave controller for angular velocity control for each propeller. The aim of the control is to stabilize coordinates of the helicopter [10].

The controllers are used as quasi-redundant components within the presented networked cascade control system (further only NCCS). They use the same control algorithm (propeller's angular velocity control) but with different input data (set point, system output, etc.)

Hence, in case of failure, one of them could retransmit all the required data to another one, whereas pre-programmed control algorithm should compute the actuating value. Thus, the failed controller is replaced by a second one which starts to compute the actuating value.

Other quasi-redundant parts of this control structure are networks (Fig. 3.6). As in the case of controllers, one of the networks can compensate another one after a system reconfiguration. Usually, two networks are primary designed due to reduction amount of transmitted data. However, in case of network failure all data could be retransmitted through the second one.

The described approach for subsystem's failure compensation by using the shared redundancy requires a logical reconfiguration of the NCCS. Thus, in case of failure the hardware configuration is non-touched but communication ways must be changed in order to transmit the data to a non-failed component or through a non-failed network.

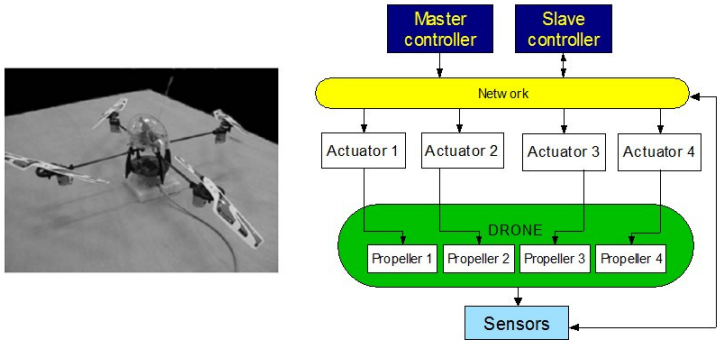


FIG. 3.5. *Cascade control structure of a mini-helicopter with one network*

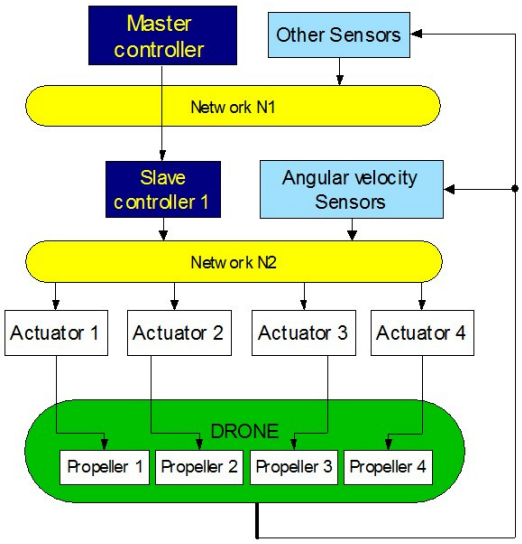


FIG. 3.6. *Cascade control structure of a mini-helicopter with two networks*

4. Simulation and results. All the presented networked control architectures (Fig. 3.5, 3.6) were modelled by using Petri nets. This tool was chosen thanks to its ability to model different types of complex systems and dependencies within them. To provide the reliability analysis, the Monte Carlo simulation (further only MCS) method was used. The multiple simulations of the modelled architecture [1] are provided to obtain the reliability behavior of the basic two quasi-redundant components (for example two controllers in CCS structure).

Model of the system covers the simulation of the random events of the basic components of the system such as sensors, controllers and actuators as well as the network's random failures. Software used for model preparation is CPN Tools which allow multiple simulation of the model in order to obtain statistically representative sample of the necessary data to determine the reliability behavior of the studied model.

As it was mentioned, the simulation of the simple two quasi-redundant components with all considered changes of the failure rate (single, linear, nonlinear) was provided. Thus, new failure rate λ' of the non-failed component is computed by using equation (3.1), (3.3) and (3.4).

This change could be called as single change because the component's failure rate is changed only once during the QR component's life time. Both components have equal nominal failure rate $\lambda_n = 0.001$.

Few examples of the influence of the single step change of the failure rate by the specified decrease factor d_R to the reliability behavior are shown in figure 4.1. We can see there are five curves. Two non-dashed curves show the studied system as a system with two active redundant components (thus, d_R is equal to zero—first curve from the top) and as system without redundant components (thus, the system composes of two independent

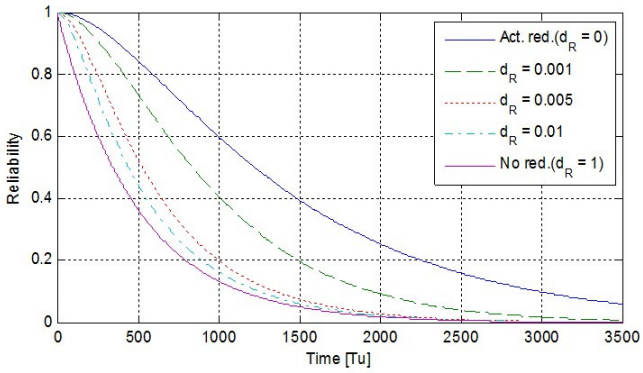


FIG. 4.1. *Influence of the increased failure rate of the component by a constant decrease factor d_R to the reliability of the system composed of two quasi-redundant components*

TABLE 4.1
MTTFF of Simulated Control Structures With Different Decrease Factors

Decrease factor—d_R	MTTFF - Drone (Fig. 3.5)	MTTFF - Drone (Fig. 3.6)
0	55(+11%)	58(+22%)
$2 * 10^{-3}$	54(+9%)	56(+17%)
10^{-2}	53(+7%)	54(+13%)
$59 * 10^{-2}$	50.5(+2%)	49(+3%)
0.999	49.6	47.6

components without redundant relation—first curve from the bottom). These two curves determine borders where the reliability of the studied system can be changed depending on the value of the decrease factor d_R.

As we can see from figure 4.1, a single increasing of the nominal failure rate λ_n of the non-failed components by the same value as was nominal failure rate λ_n up to $\lambda^{'} = 0.002$ ($d_R = 0.001$) cause a significant reduction of the reliability.

Table 4.1 show several values of the life time (parameter MTTFF) for the studied system. Each table (Table 4.1, 4.2, 4.3) shows the life time of the studied components as active redundant subsystems ($d_R = 0$) and as independent subsystems ($d_R = 0.999$). From the value of the decrease factor $d_R = 0.01$ the life time of the system significantly improves (18% and more). The results of the linear and nonlinear failure rate increasing are shown in tables 4.2 and 4.3. In all tables are noted the percentual value of the increased life time corresponding to the decrease factor.

Table 4.1 shows the MTTFF parameters of both complex mini-helicopter structures. In the first drone structure (Fig. 3.5) two quasi-redundant controllers are considered. In the second structure (Fig. 3.6) two groups of quasi-redundant subsystems are considered and simulated—the controllers and the networks.

In all simulated systems was observed the influence of the single step of the failure rate by a value specified by the decrease factor d_R. The same as in tables 4.1–4.3, there are shown the life time of system corresponding to different decrease factors 2.10^{-3}, 10^{-2}, 59.10^{-3}. We can see that increasing the component's nominal failure rate λ_n by a decrease factor equal to 59.10^{-3}, which represents approximately 59 times higher the failure rate, has a significant influence to decreasing the life time of the system. The results are a little bit better than in the case of the system without redundant components ($d_R = 0.999$), but we could see that they are almost the same.

The drone's structure composes of twenty (twenty-one—structure with two networks) components—thirteen sensors (3 gyro-meters, 3 magneto-meters, 3 accelerometers, 4 rotors' angular velocity sensors), two controllers, four actuators and one (two) networks. Due to the high ratio of independent components and shared redundant components within the drone's structure (18 independent and 2 quasi-redundant—Fig. 3.5) there is a difference between life times for minimal and maximal d_R is significantly smaller (about 11% and 22%) than in the case of a basic two components subsystem (Table 4.1, 4.2, 4.3).

The Mean Time Before First system's Failure is significantly longer in the case of a basic two component

TABLE 4.2
MTTFF of the Two Quasi-Redundant With Single Step Change of the Failure Rate

$\lambda_n = 10^{-3}$	Act. red. $d_R = 0$ ($\lambda' = 10^{-3}$)	$d_R = 0.001$ ($\lambda' = 0.002$)	$d_R = 0.005$ ($\lambda' = 0.006$)	$d_R = 0.01$ ($\lambda' = 0.011$)	$d_R = 0.1$ ($\lambda' = 0.101$)
MTTFF[Tu]	1503 (+ 300%)	1002 (+200%)	667(+34%)	589(+18%)	509(+2%)
$\lambda_n = 10^{-3}$	No red. $d_R = 0.999$ ($\lambda' = 1$)				
MTTFF[Tu]	499				

TABLE 4.3
MTTFF of the Two Quasi-Redundant With Linear Increasing of the Failure Rate

$\lambda_n = 10^{-3}$	Act. red. ($d_R = 0$	$d_R = 10^{-3}$	$d_R = 10^{-2}$	$d_R = 10^{-1}$	No redundancy
MTTFF[Tu]	1503 (+ 300%)	1153 (+231%)	812(+63%)	611(+22%)	499

subsystem than in the drone's case. As it was mentioned above this is caused by the difference in complexity between basic and drone's NCC architecture. In case of comparison between two drones structures (Fig. 3.5, 3.6) the results are better for architecture with two networks which is composed of two quasi-redundant subsystems—controllers (Master, Slave) and networks when the decrease factor is smaller than 59.10^{-3}. The increasing of the nominal failure rate by the decrease factor greater than 59.10^{-3} significantly decreases the life time of the drone. On the other side, even if the controller loading will change its failure rate approximately ten times ($d_R = 10^{-2}$) the system's life time is about 7% longer than in the case of the system without a shared redundant approach implementation.

4.1. Reliability approximation. In previous article states we focused on the description of the dependencies among QR components and their influence to the final reliability of the systems. The aim of this research is to propose a simple analytical method which describes the reliability behavior of the shared redundant subsystems with dynamically changed failure rate. Hence, in next states we introduce an analytical equation which allows approximating the reliability of the two component system. Of course, a quasi-redundant approach is considered. Thus, a finally simple method for the dependability analysis is proposed as an extension of the common known methods for the dependability analysis. The proposed method for reliability behavior approximation supposes that both quasi-redundant components have the same or similar nominal failure rate where differences are small and could be ignored. As it was mentioned above, the system composed of two QR components is considered. In this case study, we introduce only the results for reliability approximation where a single step change of the failure rate (further only FR) is considered. This FR behavior is described in the previous part of the article (3.3) by equation 3.1. Thus, let's suppose two QR components with the nominal failure rate λ_n and define the decrease factor d_R, then the reliability $R_{2qr}(t)$ behavior of the QR subsystem composed of both components can be described as follows:

$$R_{2qr}(t) = 1 - \prod_{i=1}^{2}(1 - e^{-(\lambda_n + k_i d_R)t}) \tag{4.1}$$

where k_i is the approximated coefficient.

The parameter decrease factor d_R and approximated coefficients of equation 4.1 are shown in table 4.5. In each row of the table is shown the decrease factor with the corresponding value of the coefficients k_1 and k_2. The table shows several different values of the decrease factor whereas non-mentioned values can be easily approximated by using an appropriate method.

The maximal error of the approximation given by the parameters of the equation 4.2 is less than 1

$$R_{2\lambda_n}(t) = 1 - \prod_{i=1}^{2}(1 - e^{-(\lambda_n + \frac{d_R}{2})t}) \tag{4.2}$$

where d_R is the decrease factor and λ_n the nominal failure rate of the QR components. It is necessary to explain that the error of all the approximations converge to the highest mentioned limits (1% for table's coefficients) in the bottom part of the reliability curves where the reliability of the system is smaller than 0.4. Thus, in live period when a component replacement could be already too delayed.

TABLE 4.4
MTTFF of the Two Quasi-Redundant With Exponential Increasing of the Failure Rate

$\lambda_n = 10^{-3}$	Act. red. ($d_R = 0$)	$d_R = 10^{-3}$	$d_R = 10^{-2}$	$d_R = 10^{-1}$	No redundancy
MTTFF[Tu]	1503 (+ 300%)	902 (+80%)	676(+35%)	537(+8%)	499

TABLE 4.5
Parameters of Equation 4.1 for a Single Step FR Change

Decrease factor—d_R	k_1	k_2
λ_n	0.44	0.52
$2\lambda_n$	0.39	0.395
$3\lambda_n$	0.28	0.393
$4\lambda_n$	0.198	0.434
$5\lambda_n$	0.154	0.46
$6\lambda_n$	0.13	0.4653
$7\lambda_n$	0.11	0.46
$8\lambda_n$	0.099	0.471
$9\lambda_n$	0.09	0.46
$10\lambda_n$	0.081	0.463
$20\lambda_n$	0.0445	0.38
$30\lambda_n$	0.0296	0.377
$40\lambda_n$	0.0225	0.385
$50\lambda_n$	0.0182	0.3518
$70\lambda_n$	0.0133	0.3284
$80\lambda_n$	0.011625	0.32475
$100\lambda_n$	0.0094	0.3332

4.2. MTTF parameter approximation. Each quasi-redundant subsystem does not exceed the limits of the bound of the minimal ($MTTF_{\min}$) and maximal time life ($MTTF_{\max}$) of the quasi-redundant subsystem. The parameter $MTTF_{\max}$ represents the maximal time life of the QR subsystem which could be obtained when the conditions are equal to the conditions of the subsystem with active redundant components. Thus, the nominal failure rate of the non-failed component is not changed when its load has been increased—the case when the decrease factor is equal to zero. The lowest life time limit could be defined by the parameter $MTTF_{\min}$ which characterizes the subsystem composed of the independent components. Thus, when one of the components fails the system is considered as failed. In term of the decrease factor, it is equal to 1 or $(1 - \lambda_n)$ for a single step FR change. Let's suppose the system life time limited by the bound defined by the MTTF parameter such as $\langle MTTF_{\min}; MTTF_{\max}\rangle$. These two parameters could be found by solving the simple following equations [15]:

$$MTTF_{\min} = \int_0^{\infty} \prod_{i=1}^{n} R_i(t)dt \tag{4.3}$$

and

$$MTTF_{\max} = \int_0^{\infty} (1 - \prod_{i=1}^{n}(1 - R_i(t)))dt \tag{4.4}$$

where R_i(t) is the reliability of each component.

In the final part of the results presentation we described the life time increasing of the two component QR subsystem with regard to the life time parameter $MTTF_{min}$ whereas various values of the decrease factor d_R are considered. We consider it as a simple and fast method for life time approximation. The results are shown in table 4.5. As in the previous part of this case study, we consider only the influence of the single step increasing of the nominal failure rate to the final life time of the two components QR system characterized by its MTTF parameter. In the first line, the failure rate of the non-failed QR component characterized by the multiple of the nominal failure rate λ_n. The second line shows the corresponding MTTF parameter percentage reduction within the limits defined by the abovementioned interval of the maximal and minimal life times (MTTF). The MTTF values introduced in table 4.5 are rounded, hence the method error is about $+/-2$ for the multiple of the nominal failure rate smaller or equal to $40.\lambda_n$ (decrease factor $d_R < 40$). For higher value of the decrease

TABLE 4.6
Approximated Values of the MTTF Reduction of the Two-Component QR Subsystem With Different Single Step Change of the Nominal Failure Rate λ_n

Single step change of λ_n	2 λ_n ($d_R = \lambda_n$)	3 λ_n ($d_R = 2\lambda_n$)	4 λ_n	5 λ_n	7 λ_n	10 λ_n	20 λ_n	40 λ_n	100 λ_n
Extended $MTTF_{min}$	50%	35%	25%	20%	15%	10%	5%	2%	1%

factor, the approximated error is about $+/-1$ of values shown in the table. Thus, in the case of very similar analysis result of considered complex structures it is necessary to prepare the exact model in order to obtain a more exact MTTF parameter reduction. This method could be used for the QR subsystems with the same failure rate or for the system when difference among the nominal failure rate λ_n of the components is very small and can be ignored. In the case of a nominal FR smaller than 10^{-2}, the increased value $100.\lambda_n$ should represent approximately 0.1 whereas the error could be higher. Then, it could be useful that the value of nominal FR determined for a time interval T transforms to the greater value for a shorter time interval (unit).

5. Conclusion. The paper shows the influence of additional reliability decreasing of the quasi-redundant component to entire reliability of the studied system. The description of this dependency is getting closer to show the behavior of the system reliability when a shared redundancy approach is implemented. The results shown in tables 4.1–4.3 could be very helpful in order to approximate the life time of the quasi-redundant subsystems under different conditions of the failure rate increasing. The presented cascade control architecture is suitable for a shared redundancy approach implementation and could be applied to similar systems. For example, Steer-by-Wire control [16] of two front wheels in a car, etc. In addition the paper has shown the conventional cascade control structure within conditions of networked control systems as naturally suitable to profit from quasi-redundant subsystems as networks, controllers and potentially sensors if the physical process allows it. Despite of some constraints for using this type of control, the cascade architecture is widely used in industrial control applications. Hence, only the reconfiguration algorithm should be implemented to take profit from quasi-redundant subsystems.

The case study presented in parts 4.1 and 4.2 (results section) extends the field of common methods for reliability approximation. Equations (4.1, 4.2) are considered as simple and fast analytical method in order to evaluate the reliability of the systems which covers two-component QR subsystems with single step FR change.

The main advantages of the quasi-redundant components could be summarized as follows:

- The system is composed only of necessary components (parts) for following the primary mission of the system whereas higher system reliability is ensured without using any additional active redundant components.
- Following the first point we could suppose less number of components used for saving the control mission. Thus, the economic aspect could be significant.
- Prevention of the system's critical failure when a QR subsystem has no sufficient hardware capacities.

REFERENCES

[1] J. T. SPOONER, K., M. PASSINO, *Fault-Tolerant Control for Automated Highway Systems*, in IEEE Transactions on vehicular technology, vol. 46, no. 3, 1997, pp. 770–785.

[2] F. GUENAB, D. THEILLIOL, P. WEBER, Y.M. ZHANG, D. SAUTER, *Fault-tolerant control system design: A reconfiguration strategy based on reliability analysis under dynamic behaviour constraints*, in 6th IFAC Symposium on Fault Detection, 2006, pp. 1387–1392.

[3] J. C. LAPRIE, H. KOPETZ, A. AVIŽIENIS, *Dependability: Basic Concepts and Terminology*, Chapter 1, Springer-Verlag / Wien, ISBN: 3-211-82296-8, 1992.

[4] A. MECHRAOUI, Z. H. KHAN, J.-M. THIRIET, S. GENTIL, *Co-design for wireless networked control of an intelligent mobile robot*, in ICINCO09—International Conference on Informatics in Control, Automation and Robotics (ICINCO), Italie (2-5 July 2009), pp. 318–324 ISBN: 978-989-674-000-9.

[5] R. GHOSTINE, J.-M. THIRIET, J.-F. AUBRY, M. ROBERT, *A Framework for the Reliability Evaluation of Networked Control Systems*, in 17th IFAC World Congress, July 6–11, 2008 pp. 6833–6838.

[6] C. BROSILOW, J. BABU, *Techniques of Model-Based Control*, Prentice Hall, 2002, ch. 10.

[7] P. CASTILLO, A. DZUL, R. LOZANO, *Real-Time Stabilisation and Tracking of a Four Rotor Mini-Rotorcraft*, in IEEE Transaction on control systems technology, Vol. 12, No. 4, 2004, pp. 510–516.

[8] J. Wysocki, R. Debouk, K. Nouri, *Shared redundancy as a means of producing reliable mission critical systems*, in 2004 Annual Symposium—RAMS—Reliability and Maintainability, 2004, pp.: 376–381.
[9] M. Bebbington, C-D. Lai, R. Zitikis, *Reliability of Modules with Load Sharing Components*, in Journal of Aplied Mathematics and Decision Sciences, 2007.
[10] P. Pozsgai, W. Neher, B. Bertsche, *Models to Consider Load-Sharing in reliability Calculation and Simulation of Systems Consisting of Mechanical Components*, in IEEE—Proceedings annual reliability and maintainability symposium, 2003, pp.: 493–499.
[11] J. Galdun, J. Ligus, J-M. Thiriet, J. Sarnovsky, *Reliability increasing through networked cascade control structure—consideration of quasi-redundant subsystems*, in World IFAC Congress, Seoul, South Korea, 2008.
[12] J. Galdun, R. Ghostine, J. M. Thiriet, J. Ligus, J. Sarnovsky, *Definition and modelling of the communication architecture for the control of a helicopter-drone*, in 8th IFAC Symposium on Cost Oriented Automation, 2007.
[13] J. Ligusova, J. M. Thiriet, J. Ligus, P. Barger, *Effect of Element's Initialization in Synchronous Network Control System to Control Quality*, in RAMS/IEEE conference Annual Reliability and Maintainability Symposium, 2004.
[14] S. I. Nicolescu, *Stabilité systèmes à retard—Aspects qualitatifs sur la stabilité et la stabilisation*, Diderot multimedia, 1997.
[15] I. Stary, *Spolehlivost systému*, (in Czech), CVUT, Prague, ISBN 80-01-01756-7, 1998.
[16] G. Leen, D. Heffernan, *Expanding Automotive Electronic Systems*, in Computer IEEE, Vol. 35, 2002, pp. 88–93.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 253–264. http://www.scpe.org
ISSN 1895-1767

QINNA: A COMPONENT-BASED FRAMEWORK FOR RUNTIME SAFE RESOURCE ADAPTATION OF EMBEDDED SYSTEMS

LAURE GONNORD* AND JEAN-PHILIPPE BABAU†

Abstract. Even if hardware improvements have increased the performance of embedded systems in the last years, resource problems are still acute. The persisting problem is the constantly growing complexity of systems, which increase the need for reusable developement framework and pieces of code. In the case of PDAs and smartphones, in addition to classical needs (safety, security), developers must deal with quality of service (QoS) constraints, such as resource management.

Qinna was designed to face with these problems. In this paper, we propose a complete framework to express ressource constraints during the developpement process. We propose a component-based architecture, which generic components and algorithms, and a developpement methodology, to manage QoS issues while developing an embedded software. The obtained software is then able to automatically adapt its behaviour to the physical resources, thanks to "degraded modes". We illustrate the methodology and the use of Qinna within a case study.

Key words: component, software architecture, resource dynamic management, case study.

1. Introduction.

When faced to the problem of designing handled embedded systems, the developer must be aware of the management of limited physical resources (CPU, Memory).

In order to develop multimedia software on such systems where the quality of the resource (network, battery) can vary during use, the developer needs tools to:

- easily add/remove functionality (services) during compilation or at runtime;
- adapt component functionality to resources, namely propose "degraded" modes where resources are low;
- evaluate the software's performances: quality of provided services, consumption rate *for some scenarios.*

In this context, component-based software engineering appears as a promising solution for the development of such kinds of systems. Indeed it offers an easier way to build complex systems from base components ([9]), and the management of physical resource can be done by embedding the system calls in high level components. The main advantages thus appear to be the re-usability of code and also the flexibility of such systems.

The Qinna framework ([11, 12, 3]) was designed to handle the specification and management of resource constraints problems during the component-based system development. Variability is encoded into discrete implementation levels and links between them. Quantity of resource constraints can also be encoded. Qinna provides algorithms to ensure resource constraints and dynamically adapt the implementation levels according to resource availability *at runtime.* The main advantage of the method is then the reusability of the resource components and the generic adaptation algorithms.

In this journal paper, we propose a complete formalization of Qinna framework (algorithms and components), and as proof of concept, a case study consisting of the development of a remote viewer application with the help of Qinna's implementation in C++. In Section 2 we recall Qinna's main concepts, as introduced in [11] and formalized later in [3]. In Section 3, we give an overview of Qinna's C++ implementation, and then provide the general implementation steps to develop a resource-aware application with Qinna in Section 4. Finally we illustrate the whole framework on the viewer case study (Section 5).

2. Description of the Qinna framework.

2.1. Qinna's main concepts.

The framework designed in [11] and [12], and further formalized in [3] has the following characteristics:

- Both the application pieces of code and the resource are components. The resource services are enclosed in components like `Memory`, `CPU`, `Thread`.
- The variation of quality of the provided services are encoded by the notion of *implementation level.* The code used to provide the service is thus different according to the current implementation level.
- The link between the implementation levels is made through an explicit relation between the implementation level of the provided service and the implementation levels of the services it requires. For instance, the developer can express that a video component provides an image with highest quality when it has enough memory and sufficient bandwidth.

*Université of Lille, LIFL `Laure.Gonnord@lifl.fr`
†UBO, LISyC, Université Européenne de Bretagne `Jean.Philippe.Babau@univ-brest.fr` This work has been partially supported by the REVE project of the French National Agency for Research (ANR)

- All the calls to a "variable function" are made through an existing contract that is negotiated. This negotiation is made automatically through the Qinna components. A *contract* for a service at some objective implementation level is made only if all its requirements can be reserved at the corresponding implementation levels and also satisfy some constraints called Quality of resource constraints (QoR). If it not the case, the negotiation fails.

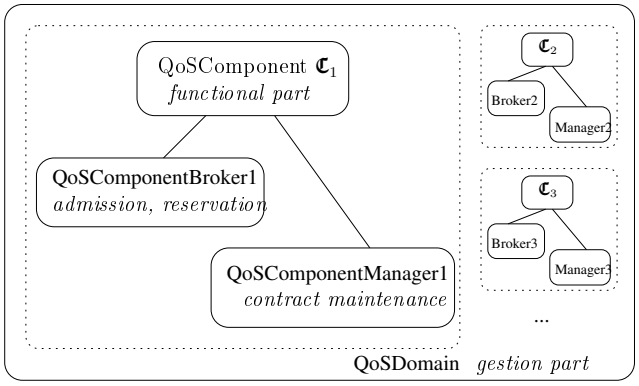


FIG. 2.1. *Architecture example*

These characteristics are implemented through new components which are illustrated in Figure 2.1: to each application component (or group of components) which provide one or more variable service Qinna associates a *QoSComponent* $\mathfrak{C}_i$. The variability of a variable service is made through the use of a corresponding `implementation level` variable. Then, two new components are introduced by Qinna to manage the resource issues of the instances of this *QoSComponent*:

- a *QoSComponentBroker* which goal is to realize the admission of a component. The Broker decides whether or not a new instance can be created, and if a service call can be performed w.r.t. the quantity of resource constraints (QoR).
- a *QoSComponentManager* which manages the adaptation for the services provided by the component. It contains a mapping table which encode the relationship between the implementation levels of each of these services and their requirements.

At last, Qinna provides a single component named *QoSDomain* for the whole architecture. It manages all the service requests inside and outside the application. The client of a service asks the Domain for reservation of some implementation level and is eventually returned a contract if all constraints are satisfied. Then, after each service request, the Domain makes an acknowledgment only of the corresponding contract is still valid.

2.2. Quantity of Resource constraints in Qinna. A Quantity of resource constraint (QRC) is a quantitative constraint on a component $\mathfrak{C}$ and the service (s_i) it proposes. QRCs are for instance formula on the total instance of a given component type, of the total amount of resource (memory, CPU) allocated to a given component. They are two types of constraints, depending on their purpose:

- Component type constraints (CTC) express properties of components of the same type and their provided services.
- Component instance constraints (CIC) express properties of a particular instance of a component.

The management of these constraints is automatically done at runtime, if the developer implements them in the following way:

- In the `QoSComponent`, for each service, implement the two functions: `testCIC` and `updateCIC`. The former decides whether or not the call to the service can be performed, and the later updates variables after the function call. In addition, there must be an initialization of the CICs formulas at the creation of each instance.
- Similarly, in the `QoSComponentBroker`, for each provided service, implement the two functions `testCTC` and `updateCTC`.

Then, Qinna maintains resource constraints at runtime through the following procedure:

- When the Broker for $\mathfrak{C}$ is created, the parameters used in `testCTC` are set.
- The creation of an instance of $\mathfrak{C}$ is made by the Broker iff $CTC_{compo}(\mathfrak{C})$ is true. During the creation, the CIC parameters are set.

- The $CIC(s_i)$ and $CTC(s_i)$ decision procedures are invoked at each function call. A negative answer to one of these decision procedures will cause the failure of the current *contract*. We will detail the notion of contract in Section 2.4.

Example The `Memory` component provides only one service `malloc`, which has only one parameter, the number of blocks to allocate. It has an integer attribute, `memory`, which denotes the global memory size and is set at the creation of each instance. We also suppose that we have no garbage collector, so the blocks are allocated only once. Figure 2.2 illustrates the difference between type and instance constraints.

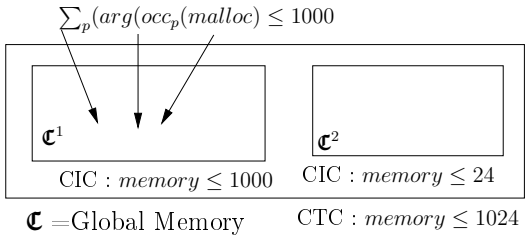


FIG. 2.2. *Type vs Instance constraints*

- CTC for $\mathfrak{C}$ = Memory: the formula $CTC_{compo}(\mathfrak{C}) \equiv \sum_j memory(\mathfrak{C}^j) \leq 1024$ expresses that the global memory quantity for the whole application is 1024 kilobytes. A new instance will not be created if its `memory` constant is set to a too big number. Then $CTC_{serv}(malloc) \equiv \sum_k arg(occ_k(malloc)) \leq 1024$ forces the calls to `malloc` stop when all the 1024 kilobytes have been allocated.
- CIC for Memory: if we want to allocate some Memory for a particular (group of) component(s), we can express similar properties in one particular instance (see $\mathfrak{C}^1$ on the Figure).

Expression of resource constraints and code generation

Qinna also provides a way to describe the resource constraints into a higher-lever language called qMEDL, a variant of MEDL event logic described in [6], and whose precise syntax and semantics is described in [3]. Roughly speaking, the logic can express boolean formulae on occurences of *events*. Atoms are of the form $Q \bowtie K$, with K constant and $\bowtie \in \{\leqslant, =, <, \ldots\}$, and Q is a quantity. The quantity are obtained by the use of auxiliary variables and calls to `value` and `time` special functions: to each event e (or $new_{\mathfrak{C}}$), $time(e)$ and $value_k(e)$ give respectively the date of the last occurrence of the event and the k^{th} argument of the function call *when it occurs*.

The Memory constraint for the whole application then can be encoded by $N \leq 1024$ where N counts the total amount of malloc's arguments: `malloc -> N:=N+value_1(malloc)`. The translation is then made by the qMEDL2 to C++ translator, and gives the following procedures (the identifiers have been changed for lisibility, `usedmem` is a local variable to count the global amount of memory used yet):

```
bool testCIC_malloc(int nbblocks){
    return (usedmem + nbblocks <= 1024)}
bool updateCIC_malloc(int nbblocks){
    usedmem = usedmem + nbblocks; }
```

2.3. QoS Linking constraints. Unlike quality of resource constraints, linking constraints express the relationship between components, in terms of quality of service. For instance, the following property is a linking constraint: " to provide the `getImages` at a "good" level of quality, the `ImageBuffer` component requires a "big" amount of memory and a "fast" network". This relationship between the different QoS of client and server services are called QoS Linking Service Constraints (QLSC).

Implementation Level To all provided services that can vary according to the desired QoS we associate an *implementation level*. This implementation level (IL) encodes which part of implementation to choose when supplying the service. These implementation levels are totally ordered for a given service. As these implementation levels are finitely many, we can restrict ourselves to the case of positive integers and suppose that implementation level 0 is the "best" level, 1 gives a lesser quality of service, and so on.

We assume that required services for a given service doesn't change according to the implementation level, that is, the call graph of a given service is always the same. However, the arguments of the required services calls may change.

Linking constraints expression Let us consider a component $\mathfrak{C}$ which provides a service s that requires r_1 and r_2 services. Qinna permits to link the different implementation levels between callers and callees. The relationship between the different implementation levels can be viewed as a function which associates to each implementation level of s an implementation level for r_1 and for r_2:

$$QLSC_s : \left| \begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N}^2 \\ IL & \longmapsto & (IL_1, IL_2) \end{array} \right.$$

This function is *statically encoded* by the developer within the application. For instance, it can easily be implemented in the QoSManager through a "mapping" table whose lines encode the tuples of linked implementation levels: $(IL_{s_1}, IL_{r_1}, IL_{r_2})$. The natural order of the lines of the table is used to determine which tuple to consider if the current negotiation fails.

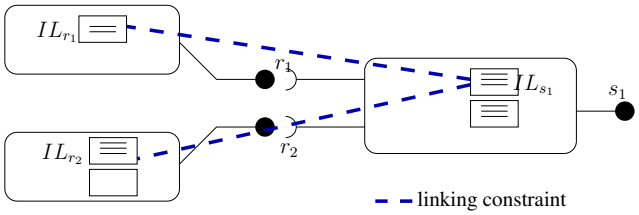


FIG. 2.3. *Implementation levels and linking constraints*

Thus, as soon as an implementation level is set for the s_1 service, the implementation levels of all required services (and all the implementation levels in the call tree) are set (Figure 2.3). This has a consequence not only on the executed code of all the involved services (and also internal functions) but also on the arguments of the service calls.

Therefore, if a user asks for the service s_1 at some implementation level, the demand may fail due to some resource constraint. That's why every demand for a service must be negotiated and the notion of contract will be accurate to implement a set of a satisfactory implementation levels for (a set of) future calls.

Implementation of linking constraints in Qinna The links between the provided QoS and the QoS of the required services are made through a table whose lines encode the tuples of linked implementation levels: $(IL_s, IL_{r_1}, IL_{r_2})$. This "mapping" table is encoded in the QoSManager. The natural order of the lines of the table is used to determine which tuple to consider if the current negotiation fails.

Now we have all the elements to define the notion of contract.

2.4. Qinna's contracts. Qinna provides the notion of *contract* to ensure both behavioral constraints (Type Constraints and Intance Constraints of services, as described in Section 2.2) and linking constraints.

When a service call is made at some implementation level, all the subservices implementation level are fixed implicitly through the linking constraints. As all the implementation levels for a same service are ordered, the objective is to find the best implementation level that is feasible (w.r.t. the behavioral constraints of all the components and service involved in the call tree).

Contract Negotiation All service calls in Qinna are made after negotiation. The user (at toplevel) of the service asks for the service at some interval of "satisfactory" implementation levels. Qinna then is able to find the best implementation level in this interval that respects all the behavioral constraints (CICs and CTCs of all the services involved in the call tree). If there is no intersection between feasible and satisfactory implementation levels, no contract is built. In the other case, a contract is made for the specific service. A contract is thus a tuple $(id, s_i, IL, [IL_{min}, IL_{max}], imp)$ denoting respectively its identifiant number, the referred service, the current implementation level, the interval of satisfactory implementation levels, and the *importance* of the contract. This last variable is used to sort the list of all current contracts and is used for degradation (see next paragraph). The importance value is statically set by the developer each time he asks for a new contract.

After contract initialization, all the service calls must respect the terms of the contract. In the other case, there will be some renegotiation.

Contract Maintenance and Degradation After each service call the decision procedure for behavioral constraints are updated. After that, a contract may not be valid anymore. As all service calls are made through the Brokers by the Domain, the Domain is automatically notified of a contract failure. In this case, the

Domain tries to degrade the contract of least importance (which may be not the same as the current one). This degradation has consequences on the resource and thus can permit other service calls inside the first contract.

Basically, degrading a contract consists in setting a lesser implementation level among the satisfactory ones, but which is still feasible. If it is not possible, the contract is stopped.

It is important to notice that contract degradation is effective only at toplevel, and thus is performed by the Domain. It means that there is no degradation of implementation level outside toplevel. That is why we only speak of contract for service at toplevel.

Use of services Each call to a service at toplevel as consequences on the contract which has been negociated for him. We suppose that a contract is made before the first invocation of the desired service. The verification could automatically be done with Qinna, but is not not yet implemented. All the notifications of failures are logged for the developer.

3. Qinna's components implementation in C++. We implemented in C++ the Qinna components and algorithms. These components are provided through classes which we detail in this section.

3.1. Qinna's components for the management of services. QoSComponent The QoSComponent class provides generic constructors and destructors, and contains a private structure to save the current implementation levels of the component provided service. All QoS components will inherit from this class.

QoSBroker The QoSBroker class contains a private structure to save the references to all the corresponding components it is responsible for. It provides the two functions `Free(QoSComponent* refQc)` and `Reserve(...)`. As `testCIC` and `updateCIC` functions signature depends of each component/service, these functions will be provided in each instance of QoSBroker.

QoSManager The QoSManager class contains all information for the service provided by its associated component. It provide the following public functions:

- `bool SetServiceInfos(int idserv, QoSComponent *compo, int nbreq, int nbmap)` initializes the manager for the *idserv* service, provided by **compo*, with *nbreq* required services and *nbmap* different implementation levels. Return `true` if successful, `false` otherwise.
- `bool AddLevQoSReq(int idserv, int lv, int irq, int lrq)` adds the tuple (lv, irq, lrq) (the lv implementation level for *idserv* is linked to the lrq implementation level for *irq* service) in the mapping table for *idserv*.
- `int Reserve(int idserv, int lv)` is used for the reservation of the *idserv* service at level *il*. It returns the local number of (sub) contract of the Manager or 0 if the reservation has failed (due to resource constraints).

QoSDomain The QoSDomain class provides functions for managing contracts at toplevel:

- `bool AddService(int service, int nbRq, int nbMp, QoSManager *qm)` adds the service *service* with *nbRq* required services and *nbMp* implementation levels, with associated manager $*qm$.
- `int Reserve(QoSComponent *compo,int ns , int lv, int imp)` is used for reservation of the service *ns* provided by the component $*compo$ at level lv and importance *imp*. it returns the number of contract (in domain) if successful, 0 otherwise.
- `bool Free(int id)` frees the contract number *id* (of domain).

ManagerContract This class provides a generic structure for a subcontract which encodes a tuple of the form $< id, lv, *rq, v >$ where *id* is the contract number, lv the current level, rq is the component that provides the service and v is a C++-vector that encode the levels of the required services. This class provides access functions to these variables and a function to change the implementation level.

DomainContract This class provides a structure for contracts at toplevel. A Domain contract is a tuple of the form $< di, i, lv, *rq >$ where di is the global identifier of the contract, $*rq$ is the manager associated to the component that provides the service, i is the local number of subcontract for the manager, and lv is the current level of the service.

Remark 1 *All services and contracts have global identifiers used in toplevel. However, it is important to notice that service and (sub) contracts have local identifiers in their respective managers.*

3.2. Basic resource components. In the call graph of one service, leaves are physical resources (Memory, CPU, Network). As all resources must be encapsulated inside components, we need to encapsulate the base functions into QoSComponents. For instance, the `Memory` component must be encoded as a wrapper around the `malloc` function, and the associated broker basically implements the CIC functions which decide if the global amount of allocated memory is reached or not.

Sometimes, the basic functions are encapsulated in higher level components. For instance, a high level library might provide a `DisplayImage` function which makes an explicit call to `malloc`, but this call is hidden by the use of the library. In this particular case, the management of basic resource functions can be done in two different but equivalent ways:

- the creation of a "phantom" Memory component which provides the two services `amalloc` (for abstract malloc) and `afree`. Each time the developer makes a call to an "implicit" resource function (*i. e.* when the called function needs a significant amount of memory, like `DisplayImage`), he has to call `Memory.amallloc`. The Qinna's C++ implementation provides some basic components like Memory, Network and CPU and their associated brokers.
- the creation of QoSComponent around the library function `DisplayImage` which is responsible (through its broker) for the global amount of "quantity of resource" used for the `DisplayImage` function.

Both solutions need a precise knowledge of the libraries functions w.r.t the resource consumption. We assume that the developer has this knowledge since he designs a resource-aware application. In our case study we used the first solution.

4. Methodology to use Qinna. We suppose that in the application all resources, including hardware resources (Memory, CPU) or software ones (viewer, buffer), are encoded by components. Here are the main steps for integrating Qinna into an existing application designed in C++:

1. **Identify the variable services** which are functions whose call may fail due to some resource reasons. They are of two types:
   - simple functions like `Memory.malloc` whose code does not vary. They have a unique implementation level.
   - "adaptive" functions whose code can vary according to implementation levels.

   The first step is thus to identify the services whose quality vary and associate to each of this services a *unique* key, and if the code vary, clearly identify the variant code through a code of the form:
   ```
   switch(implLevel)
     {
      case 0 :
          ...
     }
   ```
   where implLevel is the associated (variable) attribute of the host component for this service. We must identify which variable services are required for each provided service, and the relationship between the different implementation levels.
2. **Create Qinna components**. First, cut the source code into QoSComponents that can provide one or more QoSservices. As the QoS negotiation will only be made between QoSComponents of different types, this split will have many consequences on the QoS management. For each QoSComponentC (which inherits from the `QoSComponent` class), the designer must encode two classes: `QoSBrokerC` and `QoSManagerC` which respectively inherit from the `QoSBroker` and `QoSManager` generic classes. For the whole application, the designer will directly use the `QoSDomain` generic class.
3. **Implement Quality of Resource constraints**. These constraints are set in two different ways:
   - The type constraints (CTC) for component C implementation is composed of additional functions in $QoSBrokerC$: `initCTC` which is executed at the creation of the Broker, and which sets the decision procedures parameters ; a `testCTC` function to determine whether a new instance can be created or not ; an `updateCTC` to save modifications of the resources after the creation. For each provided QoS service s_i, we add to new functions: `testCTC(idsi)` which is executed before the call of a service and tells if the service can be done, and `updateCTC(idsi)` to be executed after the call.
   - The instance constraints (CIC) for C are also composed of three functions to encode in the $QoSComponentC$: `setCIC` to set the resources constants, `testCTC(idsi)` which is used to de-

cide if a service of identifiant `ids` can be called, and `updateCTC(idsi)` to update the resource constraints after a call to the s_i function.

4. **Implement the linking constraints**. The links between required services and provided service via implementation levels are set by the invocation of the `SetService` and `AddLevQoSReq` functions of the managers. These functions will be invoked at toplevel.
5. **Modify the main file to initialize Qinna components at toplevel**. Here are the main steps:
   - For each base resource (CPU, Memory, ...)
     (a) Invoke the constructor for the associated Broker. The constructor's arguments must contain the initialization of internal variables for type constraints (the total amount of memory for example).
     (b) Create the associated Manager with the Broker as argument.
     (c) Register the QoS services inside the Manager through the use of `SetServiceInfos` function.
     (d) Create QoSComponents instances via the `Broker.reserve(...)` function. The arguments can be a certain amount of resource used by the component.
   - For all the other QoSComponents, the required components first:
     (a) Create the associated Broker and Manager.
     (b) Set the services information.
     (c) If a service requires another service of another component, use the function `Manager.AddReq` to link the required manager. Then use `Manager.AddLevQoSReq` to set the linking constraints.
     (d) Create QoSComponent instances by invoking the corresponding reservation function (`Broker.Reserve`).
   - Create the QoSDomain and add the services that are used at toplevel (`Domain.AddService`)
   - Reserve services via the QoSDomain and save the contracts' numbers.

5. Viewer Implementation using Qinna.

5.1. Specification. Our case study is a remote viewer application whose high level specification follows:

- The system is composed of a mobile phone and a remote server. The application allows the downloading and the visualization of remote images via a wireless link.
- The remote directory is reached via a ftp connection. After connection, two buttons "Next" and "Previous" are used to display images one by one. Locally, some images are stored in a buffer. To provide a better quality of service, some images are downloaded in advance, while the oldest ones are removed from the photo memory.
- The application must manage different qualities of services for the resources: shortage of bandwidth and memory, or disconnections of the ftp server. When needed it can download images in lower quality (in size or image compression rate).
- Different storage policies are possible, and there are many parameters which can be modified; like the size of the buffer, or the number of images that are downloaded each time. We want to evaluate which policy is the best *according to a given scenario*.

We want to use Qinna for two objectives:

- the maintenance of the application with respect to the different qualities of service,
- the evaluation of the influence of the parameters, on the non-functional behavior (timing performance and resource usage).

5.2. The functional part. The functional part of the viewer is developed with Qt[1] (a C++ library which provides graphical components and implementations of the ftp protocol). Figure 5.2 describes the main parts of the standalone application. We chose to make the downloading part via the ftp protocol. The wireless part is not encoded.

- The `FtpClient` class makes a connection to an existing ftp server and has a list of all distant images. It provides a `getSome` function to enable the downloading of many files at once.
- The `ImageBuffer` class is responsible for the management of downloaded files in a local directory. As the specification says, this buffer has a limited size and different policy for downloading images. The class provides the two functions `donext` and `doprevious` which are asynchronous functions. A signal

[1] http://trolltech.com/products/qt/

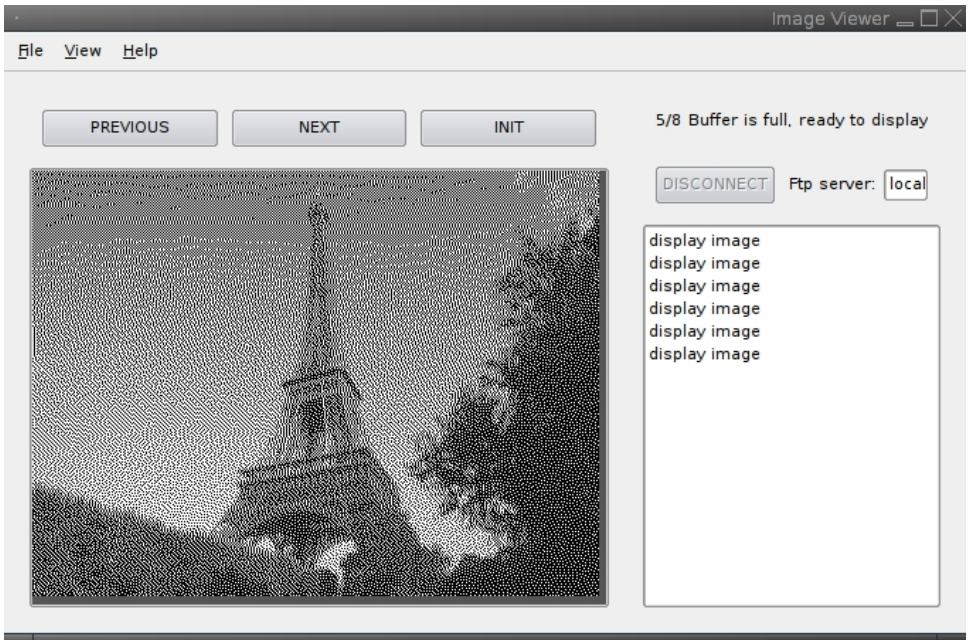


Fig. 5.1. *Screenshot of the viewer application*

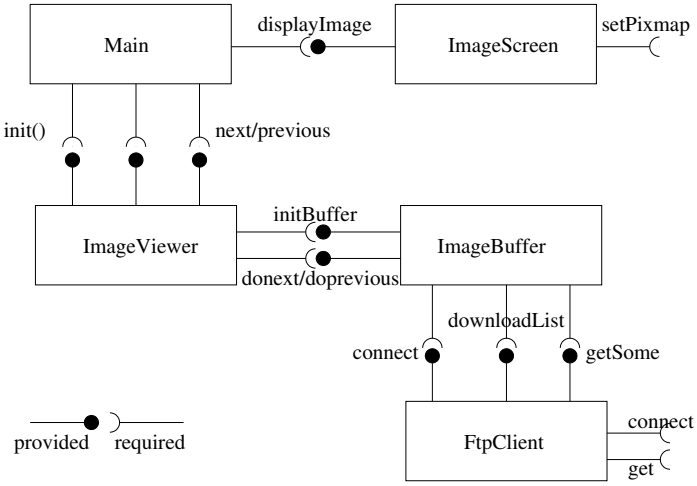


Fig. 5.2. *Functional view of the application*

is thrown if/when the desired image is ready to be displayed. It eventually downloads future images in current directory.

- The `ImageViewer` class is a high level component to make the interface between the ftp and buffer classes to the graphics components.
- The `ImageScreen` class is responsible for the display of the image in a graphic component named `QPixmap`.
- The `main` class provides all the graphics components for the Graphical User Interface.

5.3. Integration of Qinna. Now that we have the functional part of the application, we add the following resource components: Memory, and Network which are QoSComponents that provide variable services. We only focus on these two basic resources. The Network component is only linked to the FtpClient, whereas Memory will be shared between all components. For Memory, the only variable service is `amalloc` which can fail if the global amount of dedicated memory is reached ; this function has only one implementation level. For Network, the provided function `get` can fail if there is too much activity on network (notion of bandwidth).

Then we follow the above methodology in the particular case of our remote viewer.

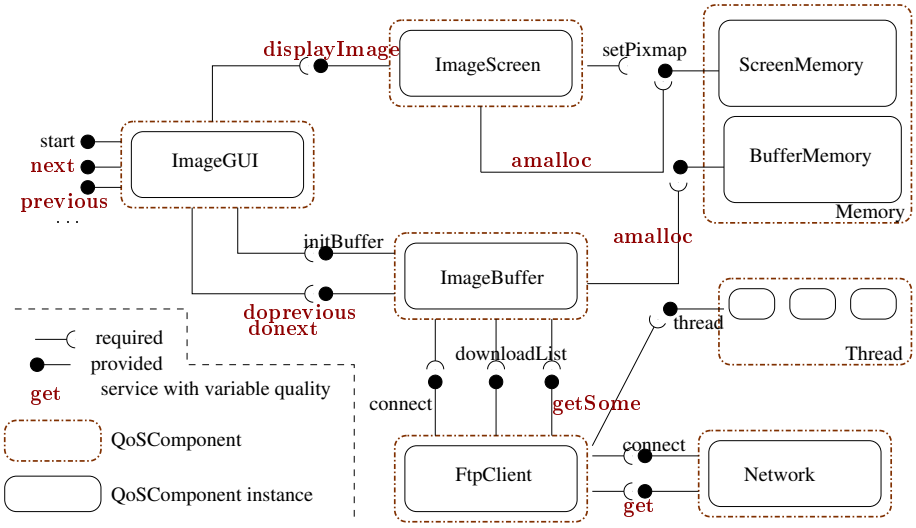


FIG. 5.3. *Architecture example*

Identification of the variable services (step 1)

Now as the variable services for low level components have been identified, we list the following adaptive services for the functional part:

- `ImageScreen.displayImage` varies among memory, it has three implementation levels which correspond to the quality of the displayed image. We add calls to `Memory.amalloc` function to simulate the use of Memory.
- `Ftpclient.getsome`'s implementation varies among available memory and the current bandwidth of network. If there is not enough memory or network, it adapts the policy of the downloads. It has three implementation levels. We add calls to `Network.bandwidth` to simulate the network resources that are needed to download files.
- `ImageBuffer.donext/previous` varies among available memory: if there is not enough memory the image is saved with high compression.

Creation of the QoSComponents (step 2)

The resource components are QoSComponents. Then, the three components `ImageScreen`, `FtpClient` and `ImageBuffer` are QoSComponents which provide each one variable service. `Imageviewer` and `Main` are QoSComponents as well. Figure 5.3 represents now the structure of the application at this step.

For the sake of simplicity, we only share Memory into two parts, a part for `ImageBuffer` and the other part for `imageBuffer`. That means that each of these components have their own amount of memory.

Resource constraints (steps 3 and 4)

The quantity of resource constraints we have fixed are classical ones (bounds for the memory instances, unique instantiation for the `imageScreen` component, no more than 80 percent of bandwidth for the ftpClient, etc). The QLSC are very similar to those described in [11] for a videogame application. Here we show how we have implemented some of these constraints in our application.

- *Quantity of resource constraints* The `imageScreen` component is responsible for the unique service `display_image` (display the image on the graphic video widget). Here are some behavioral constraints we implemented for this component:
  - There is only one instance of the component once.
  - The display function can only display images with size lesser or equal to $1200 * 800$.
  - There is only one call to the display function once.

  These type constraints are easily implemented in the associated Broker (`imageScreenBroker`) in the following way: the constraint "maximum of instance" requires two private attributes `nbinstance` and `nbinstancemax` which are declared and initialized at the creation of the Broker with values 0 and 1. Then the reservation of a new `imageScreen` by the Broker is done after checking whether or not $nbinstance + 1 \leq nbinstancemax$. If all checks are true, it reserves the instance and increments `nbinstance`.

The checking of memory is done by setting the global amount of memory for `ImageBuffer` and `imageBuffer` in local variables which are set to 0 at the beginning of each contract, and updated each time the function `amalloc` is called.
These constraints are rather simple but we can imagine more complex ones, provided they can be checked with bounded complexity (this is a constraint coming from the fact the Qinna components will also be embedded).

- *QoS Linking constraints*
To illustrate the difference between quality of resource constraints and linking constraints, we show here the constraints for the `FtpClient.getSome`:
  - The implementation level 0 corresponds to 3 successive downloads with the `Network.get` function. The function has a unique implementation level but each call to it is made with 60 as argument, to model the fact it requires 60% of the total bandwidth. These three calls are made through the use of the `Thread.thread` with implementation level 0 (quick thread, no active wait).
  - The implementation level 1 corresponds to 2 calls to the `get` function with 40% of bandwidth each time. These two calls are made through the use of the `Thread.thread` with implementation level 1 (middle thread, few active wait).
  - The implementation level 2 corresponds to 1 call to the `get` function with 20% bandwidth. This call is made through the use of the `Thread.thread` with implementation level 2 (more active wait).

Thus if the available bandwidth is too low, a negotiation or an existing contract will fail because of the resource constraints. The creation of the contract may fail because a thread cannot be provided at the desired implementation level.

Modification of toplevel (step 5) This part is straightforward. The only choices we have to make are the relative amount of resource (Memory, Network) which are allocated to each QoSComponents. The test scenario is detailed in section 5.5.

5.4. Some statistics. The viewer is written in 4350 lines of code, the functional part taking roughly 1800 lines. The other lines are Qinna's generic components (1650 loc.), 600 lines of code for the new components (imagescreenBroker, imageScreenManager *etc.*) and 300 lines of code for the test scenarios. The binary is also much bigger 4.7Mbytes versus 2Mbytes without Qinna.

Thus Qinna is costly, but all the supplementary lines of code do not need to be rewritten, because:

- Generic Qinna components, algorithms, and the basic resource components are provided with Qinna.
- The decision functions for Quality of service constraints could be automatically generated or be provided as a "library of common constraints".
- The initialization at toplevel could be computed-aided through user-friendly tables.

We think that the cost of Qinna in terms of binary code can be strongly reduced by avoiding the existing redundancy in our current implementation.

Moreover, Qinna's implementation can be viewed as a prototype to evaluate the resource use and the quality of service management. After a preliminary phase with the whole implementation used to find the best linking constraints, we can imagine an optimized compilation through glue code which neither includes brokers nor managers.

5.5. Results. We realized a scenario with a new component whose only objective is to use the basic resources Memory and Network. This `TestC` component provides only the `foobar` function at toplevel. This function has two implementation levels, and requires two functions: `ScreenMemory.amalloc` and `Network.get`. The whole application provides four functions at toplevel: `TestC.foobar`, `ImageViewer.donext` (and `doprevious`) and `ImageScreen.displayimage`. Three contracts are negotiated, in the following importance order: `foobar` first, then `donext` and `doprevious`, then `displayimage`. We made the three contracts and download and visualize images at the highest qualities, but at some point the foobar function causes the degradation of the contract for displayimage, and the images are then shown in a degraded version, like the Eiffel tower on Figure 5.1.

The gap between the characteristics of the contract and the effective resource usage can be make through the use of log functions provided by the Qinna implementation. Figure 5.4 shows for instance the memory usage for another played scenario.

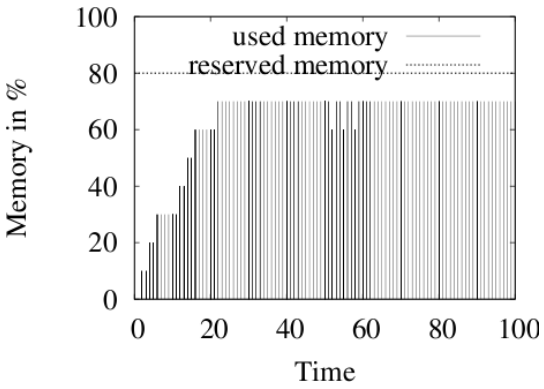


FIG. 5.4. *Memory use*

6. Related works. Other works also propose to use a development framework to handle resource variability. In [10] and [6], the author propose a model-based framework for developping self-adaptative programs. This approach uses high-level specifications based on temporal logic formula to generate program monitors. At runtime, these monitors catch the system events and activates the reconfiguration. This approach is similar to us except that it mainly deals with hybrid automata and there is no notion of contract degradation nor generic algorithm for negociation.

The expression and maintenance of resource constraints is also considered as a fundamental issue, so much work deals with this subject. In [5], the author use a probabilistic approach to evaluate the resource consumed by the program paths. Some other works in the domain of verification try to prove conformance of one program to some specification: in [7], for instance, the authors use synchronous observers to encode and verify logical time contracts. At last, the QML language ([2], [1]) is now well used to express QoS properties. This last approach is complementary to our one since it provides a language which could be compiled into source code for QoSComponents or Brokers.

7. Conclusion and future work. In this paper, we have presented a case study using the software architecture Qinna which was designed to handle resource constraints during the development and the execution of embedded programs. We focused mainly on the development part, by giving a general development scheme to use Qinna, and illustrating it on a case study. The resulting application is a resource-aware application, whose resources constraints are guaranteed at runtime, and whose adaptation to variability of service is automatically done by the Qinna components, through the notion of contracts. At last, we are able to evaluate at runtime the threshold between contractualised resource and the real amount of resource effectively used.

This work has shown the effectivity of Qinna with respect to the programming effort, and the performance of the modified application.

Future work include some improvements of Qinna's C++ components, mainly on data structures, in order to decrease the global cost of Qinna in terms of binary size, and more specific and detailed resource components, in order to better fit to the platform specifications. Integrating Qinna into a model driven development tools, such as Gaspard ([8]), can be a way to improve this efficiency.

From the theoretical point of view, there is also a need for a way to manage the linking constraints. The developer has still to link the implementation levels of required and provided services, and the order between all implementations levels is fixed by him as well. The tuning of all these links can only be done though simulation yet. We think that some methods like controller synthesis ([4]) could be used to discover the/a optimal order and linking relations w.r.t. some constraints such as "minimal variability", "best reactivity" *etc.*.

Finally, some theoretical work would be necessary in order to use Qinna as a prediction tool, and provide an efficient compilation into "glue code".

REFERENCES

[1] S. FRØLUND AND J. KOISTINEN, *Qml : A language for quality of service specification*, tech. rep., HPL-98-10, 1998.
[2] ———, *Quality of services specification in distributed object systems design*, in Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems (COOTS), Berkeley, CA, USA, 1998, USENIX Association.

[3] L. Gonnord and J.-P. Babau, *Quantity of Resource Properties Expression and Runtime Assurance for Embedded Systems*, in ACS/IEEE International Conference on Computer Systems and Applications, AICCSA'09, Rabbat, Morocco, May 2009, pp. 428–435.

[4] F. M. K. Altisen, A. Clodic and E. Rutten, *Using controller synthesis to build property-enforcing layers*, in European Symposium on Programming (ESOP), April 2003.

[5] H. Koziolek and V. Firus, *Parametric Performance Contracts: Non-Markovian Loop Modelling and an Experimental Evaluation*, in Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA), Electronical Notes in Computer Science, Vienna, Austria, 2006.

[6] I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan, *Runtime assurance based on formal specifications*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (IPDPS'99), 1999.

[7] F. Maraninchi and L. Morel, *Logical-time contracts for reactive embedded components*, in 30th EUROMICRO Conference on Component-Based Software Engineering Track, ECBSE'04, Rennes, France, Aug. 2004.

[8] I.-R. Quadri, S. Meftali, and J.-L. Dekeyser, *An mde approach for implementing partial dynamic reconfiguration in fpgas*, in Proceedings of the 16th International Conference on IP-Based System-on-chip, Grenoble, France, 2007.

[9] M. Sparling, *Lessons learned through six years of component-based development*, Commun. ACM, 43 (2000).

[10] L. Tan, *Model-based self-monitoring embedded systems with temporal logic specifications*, in Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05), 2005.

[11] J.-C. Tournier, *Qinna: une architecture à base de composants pour la gestion de la qualité de service dans les systèmes embarqués mobiles*, PhD thesis, INSA-Lyon, 2005.

[12] J.-C. Tournier, V. Olive, and J.-P. Babau, *Towards a dynamic management of QoS constraints in embedded systems*, in Workshop QoSCBSE, in conjunction with ADA'03, Toulouse, France, June 2003.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 265–275. http://www.scpe.org
ISSN 1895-1767

DEPLOYMENT OF EMBEDDED WEB SERVICES IN REAL-TIME SYSTEMS

GUIDO MORITZ, STEFFEN PRUETER, DIRK TIMMERMANN*AND FRANK GOLATOWSKI†

Abstract. Service-oriented architectures (SOA) become more and more important in networked embedded systems. The main advantages of Service-oriented architectures are a higher abstraction level and interoperability of devices. In this area, Web services have become an important standard for communication between devices. However, this upcoming technology is only available on devices with sufficient resources. Therefore, embedded devices are often excluded from the deployment of Web services due to a lack of computing power, insufficient memory space and limited bandwidth. Furthermore, embedded devices often require real-time capabilities for communication and process control. This paper presents a new table driven approach to handle real-time capable Web services communication, on embedded hardware through the Devices Profile for Web Services.

Key words: service-oriented device architecture, devices profile for Web services, Web service for devices

1. Introduction. High research efforts are made to develop cross domain communication middleware basing on architectural concepts like REST (Representational state transfer) and Service-oriented Device Architectures (SODA) [25] and on technologies like UPnP (Universal Plug and Play), JINI, and DPWS (Devices Profile for Web Services). While UPnP, DLNA and related technologies are established in networked home and small office environments, DPWS is widely used in the automation industry at device level [26] and it has been shown that they are also applicable for Enterprise integration [24, 27]. Barisic et al. [33] outline the potential of SOA to become a key factor in embedded software development. Embedded development process can be improved significantly if the SOA paradigm is used in each development stage. However, to make this happen it is necessary to establish the grounding for deeply embedded systems and real-time system

Besides the advantages of SODA, additional resources are required to host a necessary software stack. There are SODA toolkits available for resource-constrained devices like UPnP stacks [28] or DPWS toolkits [8, 9]. However, additional effort is necessary for deployment on deeply embedded devices and especially for embedded real-time systems. Deeply embedded devices are small microcontrollers with only a few kB of memory and RAM (e.g. MSP430, ARM7). These devices cannot be applied with comprehensive operating systems. But they are essential because as they combine price, low power properties, size and build-in hardware modules.

This work presents a new approach, which can be applied to deeply embedded devices and serve real-time and specification compliant DPWS communication.

2. Services in Device Controlling Systems. The World Wide Web Consortium (W3C) specifies the Web services standard [13]. UPnP is a popular specification in the home domain. Due to the lack of security mechanisms and the missing service proxy it is limited to small networks (see [2]). Furthermore, UPnP based communication scales not with the arising high number of future coming wireless smart cooperating objects due to its usage of Simple Service Discovery Protocol (SSDP) for device discovery at run-time. Web services are already widely used in large networks and the internet for business processes and server-to-server communication mainly. This client-to-server interaction uses SOAP [12] for the transport layer and Extensible Markup Language (XML) for the data representation [1, 15]. On the other hand, the Web services protocols need much computing power and memory, in order to enable a device-to-device communication with more constraint resources as servers. Therefore, a consortium lead by Microsoft has defined the Devices Profile for Web services (DPWS) [4]. DPWS uses several Web services protocols, while keeping aspect of resource constraint devices. In comparison to standard Web services, DPWS is able to discover devices at run time dynamically based on WS-Discovery, WS-MetadataExchange and WS-Transfer, without a global service registry (UDDI). The included WS-Eventing [6] specification also enables clients to subscribe for events on a device to get notified by state changes. Thus, pull messaging is avoided in favor of push messaging, which is a significant advantage for resource constraint devices and networks. DPWS is integrated in Microsofts operating systems Windows Vista and Windows 7 and furthermore in miscellaneous frameworks like e.g. .net Micro Framework. Additionally, open source stacks are available [8]. In August 2008 a technical committee (TC) at OASIS was formed for the "Web Services Discovery and Web Services Devices Profile" (WS-DD) [5]. WS-DD defines a lightweight subset of the Web Services protocol suite that makes it easy to find, share, and control devices on a network. The work of this

*University of Rostock, Rostock 18051, Germany ({guido.moritz, steffen.prueter, dirk.timmermann}@uni-rostock.de)
†Center for Life Science and Automation, Rostock 18119, Germany, (frank.golatowski@celisca.de)

TC is based on the former DPWS, WS-Discovery and SOAP-over-UDP specification. In July 2009 version 1.1 of named specifications were published by the OASIS WS-DD TC. For many companies, this is the reason for developing new interfaces for their products based on these protocols.

Using service gateways is the predominating approach to bridge between different communication layers or between embedded systems and enterprise systems. Our approach aims at lowering the efforts for integration and interaction and to set on standards instead of re-develop new protocols. Buckl et al. [32] assume a data centric processing model is used in embedded systems. Authors differentiate between embedded Services (called as eServices) and Web Services. Both worlds are integrated by a service gateway. In [34] Web Services on Universal Networks (WSUN) is described. The presented SOA based platform (environment) which is composed of broker, registry and universal adaptor and is able to bridge between different SOA technologies, like Jini and DPWS. This work concentrates on interoperability between different subnets and uses DPWS to integrate devices. Some work has been done to integrate DPWS into OSGi. While UPnP support has already been standardized within OSGi, today some initial work and proposals have been done to extend OSGi with DPWS [35, 36]. Fiehe et al. [36] describe a distributed architecture which uses DPWS to extend OSGi and make OSGi a distributed system. This approach is similar to actual work on distributed OSGi inside OSGi initiative. However, there still exists the lack to integrate DPWS capable device into OSGi.

Less work has been done to bring DPWS on deeply embedded systems and sensor networks and especially real-time systems. With the new approach, presented in this paper, Web services become also available on deeply embedded devices. Both, deeply embedded devices and devices that are more powerful will be enabled to communicate and interact with each other. This substitutes the application layer proxies.

Through linking the devices to a higher level of communication, devices no longer rely on specific transfer technologies like Ethernet. All devices in an infrastructure are connected via services. This services based architecture is already used in upper layers. Services based communication becomes available on lower layers nearest to the physical tier. This allows a higher abstraction level of process structures. The first step to allow this is the creation of a SODA framework that fulfills the requirements of deeply embedded devices.

3. Requirements for a light weight SODA. High-level communication on resource constrained embedded devices can result in an overall performance degradation. In a previous paper [7] Prueter et. al presented different challenges which have to be met in order to realize DPWS communication with real-time characteristics.

Firstly, as a basis an underlying real-time operating system must exist, ensuring the scheduling of the different tasks in the right order and in specific time slots. Secondly, the physical network has to provide real-time characteristics. The major challenge in DPWS with respect to the underlying network, is the binding of DPWS and SOAP. SOAP is bound to the Hypertext Transfer Protocol (HTTP) for transmission. HTTP is bound to the Transmission Control Protocol (TCP) [10] (see Figure 3.1). The TCP-standard includes non-deterministic parts concerning a resend algorithm in case of an error. Furthermore, the Medium Access Control (MAC) to the physical tier has to grant access to the data channel for predictable time slots. For example, Ethernet cannot fulfill this requirement.

As shown in Figure 3.1, it is possible to use SOAP-over-UDP. But in accordance to the DPWS specification, a device must support at least one HTTP interface [4].

In [7] Prueter et al. Xenomai [11] is used as operating system and RTNet [14] to grant network access with real-time characteristics. RTNet relies on the User Datagram Protocol (UDP) instead of TCP and uses Time Division Multiple Access (TDMA) for Medium Access Control (MAC). The usage of UDP demands SOAP-over-UDP at the same time. At least two interfaces have to be implemented: A non real-time, DPWS compliant HTTP/TCP interface and a real-time UDP interface. The disadvantage of using a special network stack including a special MAC, also implies building up a separate network. In this network, all participating notes have to conform to the MAC and used protocols.

For deeply embedded devices, various real-time operating systems exist. FreeRTOS [21] is a mini real-time kernel for miscellaneous hardware platforms like ARM7, ARM9, AVR, x86, MSP430 etc. Unfortunately, no useful real-time network stack and operating system combination is currently available for these kinds of deeply embedded devices. Therefore, this paper concentrates on the possibilities to provide real-time characteristics in the upper layers being on the top of TCP/IP.

The binding of DPWS and TCP through HTTP causes different challenges in granting real-time characteristic for DPWS communication and is still an ongoing work in our research group. It is not possible to reach deterministic characteristics without specific real-time operating systems and network stacks. A real-time

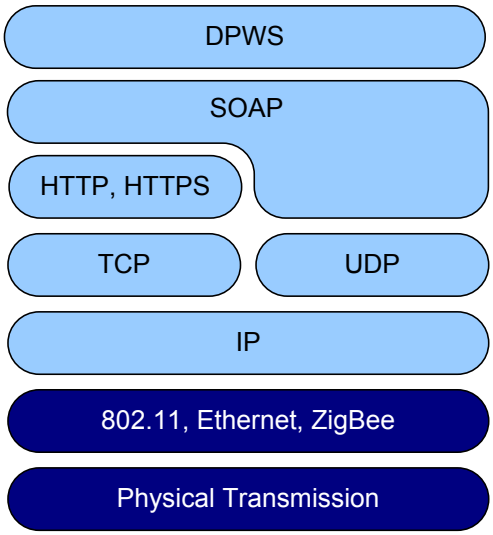


FIG. 3.1. *DPWS protocol stack*

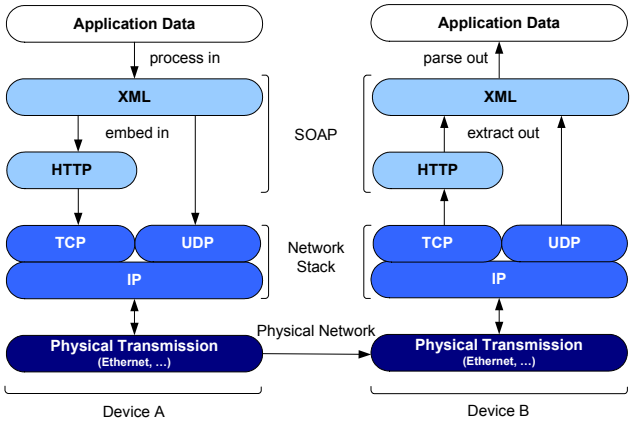


FIG. 3.2. *Modules to be implemented*

operating system grants access to peripheries for predictable time slots and execution of tasks in the right order. The arising high level communication may not interfere with the real-time process controlling. The underlying real-time operating system takes care of correct thread management and correct scheduling of the real-time and non real-time tasks. Tasks on the controller, competing with the communication, are prioritized by the operating system.

In order to provide Web services on microcontrollers, different challenges have to be met. Figure 3.2 shows the particular parts, which have to be realized.

3.1. Network Stack. The network stack, responsible for the right addressing and the way of exchanging data, is the first module, which have to be realized and meet the resource requirements. Dunkels has developed uIP and lwIP, two standard compliant TCP/IP stacks for 8 Bit controller architectures ([15, 16, 17]). uIP fulfills all minimum requirements for TCP/IP data transmissions. The major focuses are minimal code size, memory and computing power usage on the controller, without losing standard conformance. lwIP also fulfills non mandatory features of TCP/IP. Both implementations are designed to run on 8-bit architectures with and without an operating system. The differences between both stacks are shown in the following Table 3.1.

DPWS bases on WS-Discovery for automatic discovery of devices and is based on IP Multicast. Multicast applications use the connectionless and unreliable User Datagram Protocol (UDP) in order to achieve multicast communications. uIP is able to send UDP Multicast messages, but is not able to join multicast groups and receive multicast messages [17]. In contrast to uIP, the lwIP implementation supports all necessary UDP and Multicast features. The above mentioned FreeRTOS can use the lwIP stack for networking. This combines

TABLE 3.1
uIP vs. lwIP

Feature	uIP	lwIP
IP and TCP checksums	X	X
IP fragment reassembly		X
IP options		X
Multiple Interfaces		X
UDP		X
Multiple TCP connections	X	X
Variable TCP MSS	X	X
RTT estimation	X	X
TCP flow control	X	X
Sliding TCP window		X
TCP congestion control	Not needed	X
Out-of-sequence TCP data		X
TCP urgent data	X	X
Data buffered for rexmit		X

the advantages of a compatible, lightweight network stack and the usage of an embedded real-time operating system.

3.2. SOAP. Upon the network stack, HTTP communication protocol is used for transport of unicast messages. The payload is embedded in XML structures and sent via HTTP. All messages utilize the POST method of HTTP for SOAP envelope delivery. Most addressing information in the HTTP header is redundant because they are included in the SOAP message itself with a higher service abstraction. Significant HTTP header information is the Content-Length filed to identify ending of messages in the TCP data stream. Because DPWS requires a small part of the HTTP functionality only, it is not necessary to implement a full functional HTTP stack.

In contrast, the XML processing and parsing draws more attention. On deeply embedded devices, with only few kB of memory, the code size and the RAM usage have to be reduced. The WS-Discovery and WS-Metadata messages exceed the Maximum Transmission Unit (MTU) of most network technologies, including Ethernet. This supports the decision for lwIP in favour of uIP. The uIP implementation only uses one single global buffer for sending and receiving messages. The application first has to process the message in the buffer, before it can be overwritten [17]. In case of a complete XML message, the whole file has to be available before a useful parsing can be processed. Additional, computing power is restricted to resource constrained devices. With respect to the overall performance of the communication task, it is difficult to work through and parse the whole message as a nested XML file. Therefore, our research group has developed and implemented a new approach to handle HTTP and XML analysis. This new approach is described in the next section.

4. New Table Driven Approach. A complete implementation of SODA for deeply embedded systems, like wireless sensor network nodes with limited processing power and memory, is a significant challenge. All modules that are mentioned in section III like network stack, SOAP, HTTP and DPWS have to be implemented.

Due to dedicated characteristics and functionalities of sensors and actors in resource constraint environments, most of the exchanged information are discovery messages for the lose coupling of the devices during run-time and basic service invocations with non-complex data types. We have analyzed different setups with DPWS compliant implementations to identify which parts of DPWS could be omitted or adopted to reduce necessary resources. In most scenarios, only few types of messages have to be processed. After discovery and metadata exchange, the devices and their addresses are known and the services can be invoked. Only a few parts change within the exchanged messages. Major parts of the messages stay unchanged. Every time a service is called, almost the same message has to be parsed and almost the same message has to be build.

With all exchanged messages from the analysis of different scenarios, tables can be generated. The tables contain all appropriated incoming and outgoing messages. The new implemented table driven approach is able to response every request by referring to these tables.

This new table driven implementation is not based on SOAP and HTTP. Instead, we are using an approach basing on a simple string comparison of incoming messages in this new implementation. The SOAP-Envelope

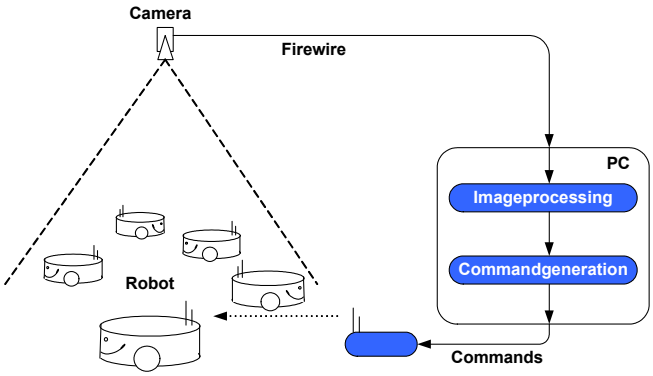


Fig. 5.1. *Mobile Robot Scenario*

containing the service invocations are simple structured and there is no need for complex parsing of messaging including e.g. heavy weight XML namespace support. The messages are interpreted as simple text messages and not as SOAP Envelopes being embedded in HTTP. The relevance of the received strings from HTTP and SOAP protocols are unknown for the table driven device. Certainly, the table driven device can analyze the incoming requests and filter required information. The device is able to send specific response with the correctly adapted dynamic changing sections. The overhead for parsing and building always the same message is reduced by this approach. Thereby memory usage and computation time are decreased in comparison to a traditional implementation.

With respect to a real-time capable communication, the treatment of the messages as strings and not as specific protocols is significant. The parsing as a string is independent of the depth of the nesting of XML structures and defined by the length of the SOAP-Envelope only. The necessary time, to parse the message as a string, is predictable. XML Schema, which is required by DPWS, cannot fulfill these requirements by default.

5. Mobile Robot Scenario. We verified our solution in a real world scenario. An external PC and an overhead camera control a team of five autonomous robots. The robots are coordinated via DPWS interfaces. The robots receive commands from a central server. The commands have to be executed in predicted timeslots to prevent collisions and enable accurate movement of the robots. The whole setup is shown in Figure 5.1.

The team behavior of the robots is controlled by a central server which uses one or more cameras mounted above the ground. Image processing software on the PC extracts the position of all robots in the field. On the PC even the commands for the robots are calculated. These commands consist of global coordinates of the robot positions and the target positions. These commands are sent with a high transmission rate to the robots. The robots use global coordinates to update their own local and imprecise coordinate tracking. The robots need this global updates in regular periods, otherwise a correct controlling cannot be granted. These real-time requirements for controlling the robots with a parallel running communication system make the robot scenario an ideal test bench for our implementations.

5.1. Robot Hardware. To control the robots we use two controller boards alternatively: an embedded Linux board and an ARM7 controller board. The embedded Linux board is the Cool Mote Master (CMM) from LiPPERT. It is equipped with an AMD Alchemy AU 1550 processor [19]. This board is designed as a gateway node for sensor networks. The CMM is already equipped with an 802.15.4 compliant transceiver. We have extended the board with additional Bluetooth and Wi-Fi (IEEE 802.11) interfaces [20]. Thereby, the board has three different radio technologies for networking beside Ethernet.

The ARM7 board is a SAM7-EX256 evaluation board from Olimex [23]. This board is applied with an Atmel ARM7 controller with 256 kB memory and 64 kB RAM. The board already provides an Ethernet interface, which was used for testing. The controller is running with a clock rate of 55MHz. It is possible to schedule the lwIP stack and the implemented table driven device in different prioritized tasks with the help of FreeRTOS.

The implementations are evaluated on a standard PC and on these boards. An overview of used hardware is provided in Table 5.1. The network devices are configured in a way, that all of them can handle IP traffic.

6. Implementation. Our research group has implemented the WS4D-gSOAP toolkit [8]. This is a software solution, which includes a DPWS stack and software tools for creating of own Web services based on

TABLE 5.1
Used hardware for testing the new table driven approach

	x86 PC	CMM	SAM-7
CPU	Intel Pentium 4	Alchemy AU 1550	Atmel ARM7
Clock Rate	3,4 GHz	500 MHz	55 MHz
ROM	500 GB	512 MB	256 kB
RAM	1024 MB	256 MB	64 kB
Operating System	Linux (2.6.24/Ubuntu)	Linux (2.6.17/Debian)	FreeRTOS 5.0
Network interfaces	Ethernet	Ethernet, 802.11g, 802.15.4, Bluetooth	Ethernet

DPWS. This toolkit uses gSOAP [22] for SOAP functionalities and extends gSOAP with an implementation of the DPWS specification. This traditional implementation will be used as benchmark for the new table driven approach.

In the first step a service is created with the existing WS4D toolkit that provides all necessary commands for the robots in our mobile robot scenario. The external PC calls a hosted service on the robots. The service is called every time when new commands have to be send to the robots. The new commands are embedded in the request. The service answers with a response, including a performance parameter of the robot.

In the second step, the exchanged messages are analyzed according to the DPWS specification. All possible outgoing and incoming messages for the mobile robot scenario are generated. In the third step, a completely new DPWS device is implemented. The structures and contents of the possible messages are deposited in the new implemented device as strings. This device does not support any dynamic SOAP or HTTP functionalities. The new table driven approach does not parse the whole incoming message as XML file. Every received message is analyzed with an elementary string compare. Thereby the type of the message is figured out. If the message type is known, the device answers with the related message. The answer is already deposited in the implemented device as a string also. In the answer, only parts required by the DPWS specification and the payload are changed. With respect to available resources of the target hardware platform, the implementation can be optimized concerning the Flash and RAM memory usage. On the one hand, the generated tables can be loaded in RAM at start time of the binary. This requires more available main memory, but can fasten the data access during run time. On the other hand, the generated tables can also be stored in non-volatile memory like Flash. This reduces footprint of the binary but may cause higher execution times during service invocation and message processing. To meet real-time requirements, the choice between both options correlates to real-time features of the underlying management of volatile and non-volatile memory access.

During the implementation of the table driven device, we have taken care that system functions are not called in critical sections. For example, the main memory management is provided by the task itself. The task allocates a pool of main memory when it is started and then organizes the main memory itself. Furthermore, the different threads for the network stack and the threads handling the messages are analyzed to be scheduled in the right order and with correct priorities.

6.1. Message Exchange. Figure 6.1 gives an overview of exchanged messages in the mobile robot scenario. When starting the device, it announces itself with a Hello SOAP Envelope. Within this message only the MessageID and the transport specific address, are dynamically and has to be adapted. Furthermore, the MessageNumber and the InstanceID has to be correct.

When a client was not started, as the device announces itself with a Hello, the client asks with a Probe for available devices. The answer is a Probe Match, where the RelatesTo has to fit to the MessageID of the Probe and the MessageID has to be dynamic. Here, also the MessageNumber and the InstanceID has to be incremented.

When the devices and their addresses are known, the client will ask for the hosted services on the device in the next step. Therefore, a GetMetadata Device is send to the hosting service, which is at least a hosted service that announces representative the available hosted services. The GetMetadata message is the first one that is sent via HTTP. Within the HTTP header, the Content-Length header field, the length of the message, and the IP address has to be adopted. The address only has to be changed, if it was not known at compile time. This applies to all IP addresses in the scenario. In the GetMetadata Device message SOAP-Envelope, the To XML tag has to match to the address of the device, detected through the Probe. The device answers with a GetMetadata Device Response message. In this message the RelatesTo has to match the MessageID of the GetMetadata Device.

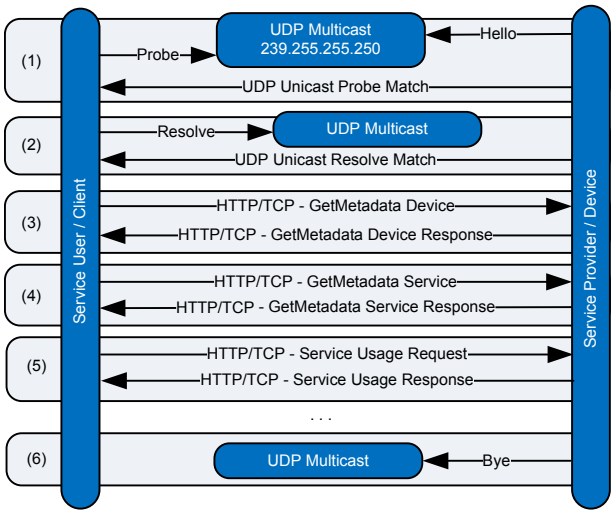


Fig. 6.1. *Message Exchange*

When the client knows available hosted services, the specific hosted service, that the client is looking for, is asked for the usage interface with a GetMetadata Service. The GetMetadata Service Response refers to the GetMetadata Service through the RelatesTo XML tag.

After the metadata exchange is complete, the client knows how to interact with the specific service and the service usage starts. The client invokes the service with a message, where To and MessageID has to be correct. In the Service Usage Request, the coordinates of the mobile robot scenario are integrated. The service answers with the Service Usage Response. Therein, the reference to the request is given through the RelatesTo tag. In our special mobile robot scenario, the response also contains the ProcessingTime tag. In this section, the service informs the service user about the time, the application needs to process the new coordinates and is a performance parameter for the mobile robot.

An overview about the dynamic parts of the different messages is given in Table 6.1. The overall size for the exchanged messages is 12.839 Bytes. The overall number of Bytes that can change is 588. Only 4.6% of the overall exchanged bytes are dynamic in the mobile robot scenario.

6.2. Devices Footprint. The memory optimized WS4D toolkit implementation of the DPWS device needs 360 kB of disk space when compiled for Linux on a x86 architecture. The table driven device implementation has a 16 kB footprint when compiled for a standard x86 PC running with Linux. Both versions do not contain networking stacks in these x86 implementations. Both implementations for an x86 PC running with Linux are using the BSD Socket API and corresponding network stacks included in Linux to handle the network traffic. The same implementation of the new table driven approach ported to the SAM7-EX256 board running with FreeRTOS 5.0 has a 13 kB footprint without network stack and interface drivers. As network stack the independent lwIP stack in Version 1.3 is applied to the board. Therefore, the stack was ported to FreeRTOS 5.0.

The required disk space for the different parts on the SAM7 board is shown in Table 6.2. The overall memory being used on the board, including FreeRTOS, lwIP and the device needs 146 kB.

The heap and stack usages of both implementations are given in Table 6.3. The maximum stack and heap usage of the table driven approach is much lower, because exchanged messages and their sizes are known at compile-time and no non-required memory has to be allocated during run-time. Due to the soft resource requirements of the used hardware platforms, the implementation of the table driven approach is still not full optimized concerning heap and sack usage. It depends on the specific scenario, if the message tables are kept into RAM during run-time or are loaded separately into RAM from non-volatile memory like flash on demand. Keeping the contents of the tables in flash reduces heap and stack usage, but may cause a gain in responds time due to higher access time to flash compared to RAM. For highly energy constrained devices with flash memory for data storage on external hardware modules, swapping the tables to flash can have an influence on the overall power consumption also. The flash hardware component can be switched of while keeping the tables without consuming any energy, but have to be switched on every time when access to the tables is required. Thus, depending on the specific scenario, a hybrid solution for table storage might be optimal. Often required tables

TABLE 6.1
Overview Exchanged Messages

Message Type	Changing parts	Dynamic Bytes
Hello	MessageID	36
	XAddrs (IP)	max. 17
	AppSequence MessageNumber	approx. 2
	AppSequence InstanceId	10
Probe	MessageID	36
Probe Match	MessageID	36
	RelatesTo	36
	AppSequence MessageNumber	approx. 2
	AppSequence InstanceId	10
GetMetada Device	HTTP Content-Length	max. 5
	HTTP Host	175, max. c.f. [10]
	MessageID	36
	To	36
GetMetadata Device Response	HTTP Content-Length	max. 5
	RelatesTo	36
	Address	175, max. c.f. [10]
GetMetadata Service	HTTP Content-Length	max. 5
	HTTP Host	175, max. c.f. [10]
	MessageID	36
	To	175, max. c.f. [10]
GetMetadata Service Response	HTTP Content-Length	max. 5
	RelatesTo	36
Service Invocation Request	HTTP Content-Length	max. 5
	HTTP Host	175, max. c.f. [10]
	MessageID	36
	To	36
	Payload	16
Service Invocation Response	HTTP Content-Length	max. 5
	RelatesTo	36
	Payload	3

TABLE 6.2
Footprint of SAM7 Implementation with FreeRTOS and lwIP

Module	Footprint
static DPWS device	13 kB
lwIP 1.3	77 kB
FreeRTOS including Debug Tasks	56 kB

and content are kept into RAM with low access times and no additional energy consumption, while infrequent used tables are kept into flash to reduce heap and stack usage while run-time.

6.3. Time Responds. Also some timing measurements have been done in order to have an objective comparison for the new static approach. Therefore, the round trip time was measured that is required from sending the message to receiving the response on the client side. Through this method the overall performance and the maximum number of service invocations per second can be determined which can be served.

These measurements are done for a standard x86 PC and the SAM7 board. On both devices a 100 MB/s Ethernet interface is applied, which has been used for the measurements. On the SAM7 boards, an independent thread simulated an additional CPU load. This CPU load thread was scheduled with different priorities. As requesting client a standard PC (2x3,5 GHz with 1 GB RAM) was used in all cases.

The following Table 6.3 shows the times measured for the different implementations of DPWS server/device. The values are the average over 1000 requests, send back-to-back.

TABLE 6.3
Round Trip Time and Memory Usage

Device	Round Trip Time	Requests per sec	Max Heap Usage	Max Stack usage
x86 PC WS4D toolkit	1,05 ms	952	112,76 Bytes	97,64 Bytes
x86 PC Table Driven DPWS	0,9 ms	1111	8,928 Bytes	15,324 Bytes
SAM7-EX256 Table Driven DPWS	18,6 ms	53	N.A.	N.A.
SAM7-EX256 Table Driven DPWS	18,6 ms	53	N.A.	N.A.
SAM7-EX256 Table Driven DPWS	30,2 ms	33	N.A.	N.A.

On the PC, the new implemented approach provides a faster overall processing. The time responds on the real-time operating system of the SAM7 board, depend on given priorities for the different competing tasks. As long as the CPU load task has a lower priority than the DPWS and the lwIP tasks, no effect to the average times could be measured.

7. Message structure optimizations. All messages in DPWS make use of XML for data representation. The application of XML in DPWS and Web Services has multiple advantages considering independency of programming language, operating system, communication channel, data representation, and character set. Certainly, XML implies a message overhead. Hence, this subsection describes several concepts for optimized data encodings of SOAP messages.

Fast Web Services [30] are using ASN to compress the XML files into a resource optimized binary representation and to overcome performance issues for complex string operations in message processing. Sandoz et al. developed a solution to convert Web Services specific XML Schema into ASN. The proposed approach reduces the size of the XML data by more than factor four and "performance is nearly 10 times that of XML literal. In other words, Fast Web Services will perform better as the size of the content increases." Nevertheless, this approach leads to isolated applications and is incompatible with DPWS devices and clients that do not support the ASN data representation.

The Efficient XML Interchange Working Group [29] develops an encoding format for XML, "that allows efficient interchange of the XML Information Set and allows effective processor implementations". The main focus is high data compression even of big and deep structures completely compliant to XML. Analyses have shown that the binary documents can be up to 90% smaller than the original XML document.

In comparison to Efficient XML and Fast Web Services, A-SOAP [31] describes concepts for XML encoding, which can be easily implemented in hardware and thereby much more energy efficient then in software. Additionally, this approach provides real-time parsing characteristics.

A-SOAP (Adaptive SOAP) uses hash functions, to encapsulate complex XML structures. Constantly recurrent XML structures are represented by hash values (see example in Figure 7.1). For the transmission only changing parts of the XML files are transmitted as proper XML tags. All tags known by sender and receiver are transmitted by using hash values. Especially A-SOAP can be integrated in DPWS and assure compliance to clients and devices which not support A-SOAP. An endpoint that cannot understand a SOAP message, responds with a SOAP Fault message. In the case when an endpoint is not A-SOAP enabled, the overhead is one additional request and one additional SOAP Fault. The sender then has to retransmit the message as a compliant XML message. Certainly, this generic A-SOAP support detection mechanism is completely DPWS compliant.

A-SOAP is a proper addition to the table driven approach. The contents of the generated tables can be attached to hash values for identification. Not the contents have to be transmitted but only the dedicated hash values. On the one hand, this reduces parsing efforts as hash values can be parsed in hardware and in software fast and easily. On the other hand, the integration of the A-SOAP approach in toolkits for the implementation of the table driven approach reduces footprints and memory consumption significantly. If possible communication partners are already known at compile-time and thus all clients invoking the table driven device comply the DPWS extended with A-SOAP functionalities, the tables might be omitted completely in the binaries. Only

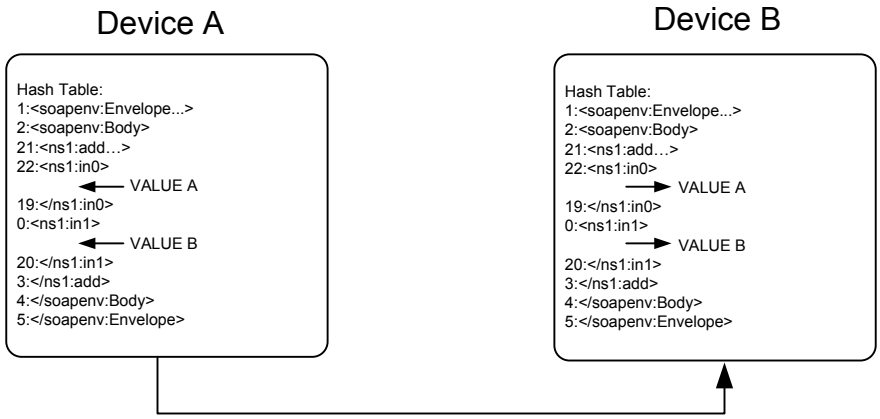


FIG. 7.1. *A-SOAP*

the associated hash values have to be included. This results in real-time DPWS based communication even for highly resource constrained platforms.

The disadvantage of A-SAOP is that it recently was granted as patent. Hence, there is no proposal for a data compression to apply DPWS in WSNW.

8. Conclusion. The new table driven approach allows the usage of Web services on deeply embedded devices. Furthermore, the implemented services can grant real-time capabilities. Thus, the deeply embedded devices can be integrated in enterprise service structures. The created service interfaces can be reused in different application. The connectivity between such large numbers of embedded devices normally needs proxy concepts with static structures. Now, these proxies are no longer required. The devices can be directly accessed by a high level process logic. Furthermore, the validation and certification become cheaper because of the slim implementation and reusability of the interfaces.

The measurements show that the binary size of a device can be reduced by the factor of more than 20. At the same time, the time responds can be improved. Heap and stack usages do not depend on specific dependent values but on specific message exchange patterns of dedicated scenarios Through the implementation in different threads, the time responds of the new implemented static approach is independent from other competing tasks. However, this assumes an underlying real-time operating system.

Further optimization of the footprint and dynamic memory usage are a main focuses for the future work. Future work will also research on a completely specification compliant implementation including optimized message structuring for real-time parsing.

REFERENCES

[1] W. DOSTAL, M. JECKLE, I. MELZER, AND B. ZENGLER, *Serviceorientierte Architekturen mit Web Services*, Elsevier, (2005).
[2] ELMAR ZEEB, ANDREAS BOBEK, HENDRIK BOHN, FRANK GOLATOWSKI, *Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services*, 2nd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2007), (2007), Niagara Falls, Ontario, Canada,pages 956–963.
[3] MARCO SGROI, ADAM WOLISZ, ALBERTO SANGIOVANNI-VINCENTELLI AND JAN M. RABAEY, *A Service-Based Universal Application Interface for Ad-hoc Wireless Sensor Networks (Draft)*, Unpublished article, (2003).
[4] MICROSOFT CORPORATION, *DPWS Specification*, Technical Report, http://specs.xmlsoap.org/ws/2006/02/devprof, (2006).
[5] OASIS, *Web Services Discovery and Web Services Devices Profile (WS-DD) TC*, Technical Report, http://www.oasisopen.org/committees/ws-dd/, (2009).
[6] WORLD WIDE WEB CONSORTIUM, *Web Services Eventing (WS-Eventing) Submission*, Technical report, http://www.w3.org/Submission/WS-Eventing/, (2006).
[7] STEFFEN PRUETER, GUIDO MORITZ, ELMAR ZEEB, RALF SALOMON, FRANK GOLATOWSKI, DIRK TIMMERMANN, *Applicability of Web Service Technologies to Reach Real Time Capabilities*, 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), (2008), Orlando, Florida, USA, pages 229-233.
[8] UNIVERSITY OF ROSTOCK, *DPWS-Stack WS4D*, Technical report, http://ws4d.org, (2009).
[9] SCHNEIDER ELECTRIC, *SOA4D Forge*, Technical report, https://forge.soa4d.org/, (2009).
[10] INTERNET ENGINEERING TASK FORCE, *Hypertext Transfer Protocol—HTTP/1.1 [RFC 2616]*, Technical report, http://tools.ietf.org/html/rfc2616, (1999).
[11] PHILIPPE GERUM, *Xenomai - Implementing a RTOS emulation framework on GNU/Linux*, Whitepaper, (2004).

[12] World Wide Web Consortium, *Simple Object Access Protocol Specification*, Technical report, http://www.w3.org/TR/soap/, (2008).
[13] World Wide Web Consortium, *Web Service Architecture Specification*, Technical report, http://www.w3.org/TR/ws-arch/, (2007).
[14] Kiszka, J. and Wagner, B. and Zhang, Y. and Broenink, J.F., *RTnet - A flexible Hard Real-Time Networking Framework*, 10th IEEE International Conference on Emerging Technologies and Factory Automation Volume 1, (2005) Catania, Italy, page 8.
[15] Adam Dunkels, *Full TCP/IP for 8-Bit Architectures*, International Conference On Mobile Systems, Applications And Services, (2003), San Francisco, California, pages 85-98.
[16] Adam Dunkels, *uIP*, Technical report, http://www.sics.se/~adam/uip/, (2007).
[17] Adam Dunkels, *lwIP—A Lightweight TCP/IP stack*, Technical report, http://savannah.nongnu.org/projects/lwip/, (2008).
[18] Adam Dunkels, *The uIP Embedded TCP/IP Stack*, The uIP 1.0 Reference Manual, Technical report, (2006).
[19] LiPPERT: PC solutions for rugged industrial applications, *Cool Mote Master Board*, Technical report, http://www.lippert-at.com/, (2008).
[20] Thorsten Schulz, *Evaluierung verschiedener Prozessorloesungen fuer RoboCup Roboter*, Technical report, (2006).
[21] FreeRTOS— The Standard Solution For Small Embedded Systems, http://www.freertos.org, (2008).
[22] Robert A. van Engelen, Kyle A. Gallivany, *The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks*, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), (2002) Washington, DC, USA, page 128.
[23] Olimex, SAM7-EX256 Evaluation Board, Technical report, http://www.olimex.com/dev, (2008).
[24] S. Karnouskos, O. Baecker, L. Moreira, S. de Souza, P. Spiess, *Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure*, Emerging Technologies and Factory Automation (ETFA), (2007) Patras, Greece, pages 293-300.
[25] Scott de Deugd, Randy Carroll, Kevin E. Kelly, Bill Millett, and Jeffrey Ricker, *SODA: Service-Oriented Device Architecture*, IEEE Pervasive Computing, (2006), vol. 5, no. 3, pages 94–C3.
[26] H. Bohn, A. Bobek, and F. Golatowski, *SIRENA - Service Infrastructure for Realtime Embedded Networked Devices: A service oriented framework for different domains*, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06), (2006), Washington, DC, USA, page 43.
[27] Elmar Zeeb, Steffen Prueter, Frank Golatowski, Frank Berger, *A context aware service-oriented maintenance system for the B2B sector*, 3rd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2008), (2008) Ginowan, Okinawa, Japan, pages 1381–1386.
[28] Intel, *UPnP - Technology Overview*, Technical report, http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/overview/, (2008).
[29] W3C, *Efficient XML Interchange Working Group*, Technical report, http://www.w3.org/XML/EXI/, (2009).
[30] Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M., Pelegri-Llopart, E., *Fast Web Services*, Article, (2003).
[31] Rosu, M.-C., *A-SOAP: Adaptive SOAP Message Processing and Compression*, IEEE International Conference on Web Services (ICWS2007), (2007) Salt Lake City, Utah, USA, pp.200-207.
[32] Christian Buckl, Stephan Sommer, Andreas Scholz, Alois Knoll, Alfons Kemper, Joerg Heuer, Anton Schmitt, *Services to the Field: An Approach for Resource Constrained Sensor/Actor Networks*, International Conference on Advanced Information Networking and Applications Workshops (AINAW2009), (2009) pp. 476–481.
[33] Daniel Barisic, Martin Krogmann, Guido Stromberg, Peter Schramm, *Making Embedded Software Development More Efficient with SOA*, 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW2007), (2007) vol. 1, pp. 941–946.
[34] Hyung-Jun Yim, Il-Jin Oh, Yun-Young Hwang, Kyu-Chul Lee, Kangchan Lee, and Seungyun Lee, *Design of DPWS Adaptor for Interoperability between Web Services and DPWS in Web Services on Universal Networks*, International Conference on Convergence Information Technology (ICCIT 2007), (2007) pp. 1032–1039.
[35] André Bottaro, Anne Gérodolle, Sylvain Marié, Stéphane Seyvoz, Eric Simon, *RFP 86 DPWS Discovery Base Driver*, OSGi Alliance, (2007).
[36] A. Bottaro and A. Gérodolle, *Home SOA: facing protocol heterogeneity in pervasive applications*, 5th international Conference on Pervasive Services (ICPS2008), (2008) ACM, New York, NY, pp. 73–80.
[37] Christoph Fiehe, Anna Litvina, Ingo Lueck, Oliver Dohndorf, Jens Kattwinkel, Franz-Josef Stewing, Jan Krueger, Heiko Krumm, *Location-Transparent Integration of Distributed OSGi Frameworks and Web Services*, International Conference on Advanced Information Networking and Applications Workshops, (2009) pp. 464–469.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 277–289. http://www.scpe.org
ISSN 1895-1767

TOWARDS TASK DYNAMIC RECONFIGURATION OVER ASYMMETRIC COMPUTING PLATFORMS FOR UAVS SURVEILLANCE SYSTEMS

ALÉCIO P. D. BINOTTO*, EDISON P. DE FREITAS†, MARCO A. WEHRMEISTER‡, CARLOS E. PEREIRA§, ANDRÉ STORK¶, AND TONY LARSSON‖

Abstract. High-performance platforms are required by modern applications that make use of massive calculations. Actually, low-cost and high-performance specific hardware (e.g. GPU) can be a good alternative along with CPUs, which turned to multiple cores, forming powerful heterogeneous desktop execution platforms. Therefore, self-adaptive computing is a promising paradigm as it can provide flexibility to explore different computing resources, on which heterogeneous cluster can be created to improve performance on different execution scenarios. One approach is to explore run-time tasks migration among node's hardware towards an optimal system load-balancing aiming at performance gains. This way, time requirements and its crosscutting behavior play an important role for task (re)allocation decisions. This paper presents a self-rescheduling task strategy that makes use of aspect-oriented paradigms to address non-functional application timing constraints from earlier design phases. A case study exploring Radar Image Processing tasks is presented to demonstrate the proposed approach. Simulations results for this case study are provided in the context of a surveillance system based on Unmanned Aerial Vehicles (UAVs).

Key words: reconfigurable computing, dynamic scheduling, aspect-oriented paradigm, unmanned aerial vehicles

1. Introduction. In addition to timing constraints, modern applications usually require high performance platforms to deal with distinct algorithms and massive calculations. The development of low-cost powerful and application specific hardware (e.g., GPU—Graphics Processing Unit, the Cell processor, PPU—Physics Processing Unit, DSP—Digital Signal Processor, PCICC—PCI Cryptographic Co-processor, FPGA—Field Programmable Gate Array, among others) offers several alternatives for execution platforms and application implementation, aiming at better performance, programmability and data control. The resulting heterogeneity in the execution platform can be considered as an asymmetric multi-core cluster. This cluster's processing power is intensified with the new generation of multi-core CPUs, being a challenge to program applications that use efficiently all available resources and Processing Units (PU).

In this sense, low-cost hybrid hardware architectures are becoming attractive to compose adaptable execution platforms. Thus software applications must benefit from that powerfulness. This leads to the creation of new strategies to distribute applications' workload (tasks, algorithms, or even full applications that must run concurrently) to execute in asymmetric PUs in order to better meet application's requirements, such as performance and timeliness, without loosing flexibility. Dynamic and reconfigurable load-balancing computing (by means of task allocation reconfiguration, i. e., rescheduling) is a potential paradigm for those scenarios, providing flexibility, improving efficiency, and offering simplicity to program an (balanced) application on heterogeneous and multi-core architectures. Fig. 1.1 shows such a theoretical scenario of a desktop-based platform composed of several devices.

An important step towards the usage of the above mentioned hybrid platforms is to create a real-time workload self-rescheduling framework to balance the resource usage by applications composed of different algorithms (graphics, massive mathematical calculations, sensor data processing, artificial intelligence, cryptography, etc.), executing on top of such hybrid platforms under time constraints, in order to achieve a minimal Quality of Service. In addition, it has to be predicted that during execution time, new tasks can arise and influence the whole system. In this manner, such framework must keep monitoring the tasks' performance to provide online information for a possible new allocation balance, indicating that task rescheduling may be necessary to promote a better performance for the overall current scenario.

In this paper, the focus is on the very first step in the reconfiguration framework: application requirements handling (rescheduling) in a high-level design phase. The approach is based on application requirements, like

*Fraunhofer IGD/TU Darmstadt, Darmstadt, Germany / Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil (alecio.binotto@igd.fraunhofer.de).

†School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden / Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil (edison.pignaton@hh.se).

‡Department of Automation and Systems Engineering, Federal University of Santa Catarina, Florianópolis, Brazil (marcow@das.ufsc.br).

§Electrical Engineering Department, Federal University of Rio Grande do Sul, Porto Alegre, Brazil (cpereira@ece.ufrgs.br).

¶Fraunhofer IGD/TU Darmstadt, Darmstadt, Germany (andre.stork@igd.fraunhofer.de).

‖School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden (tony.larsson@hh.se).

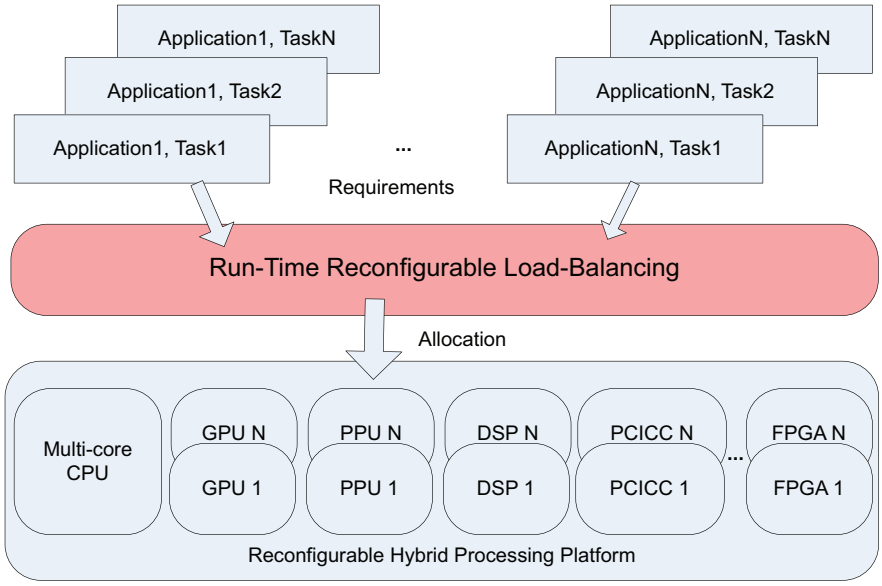


FIG. 1.1. *System overview.*

task/application deadlines, in order to find the current allocation balance that minimizes the whole application(s) execution time. This information is used by the framework to balance the computation over the execution platform. For its accomplishment, crosscutting concerns related to real-time non-functional requirements are taken into account. Handling these concerns by specific design elements called "aspects" (from aspect-oriented paradigm [11]) plays an important role for understandability and maintainability of the system design, as those concerns may influence different parts of the system in different ways. Then, based on the support offered by the aspects to monitor and control the resource usage (profiling), a strategy to assign tasks dynamically is presented, which is submitted to a run-time rescheduling when it is needed.

The paper is organized as follows. Section 2 starts with a previous work on aspects and requirements identification, modeled using UML. Section 3 follows with the dynamic workload strategy implemented by the created aspects. Composing these two concepts, Section 4 outlines an UAV surveillance system as case study, focusing on RIP (Radar Image Processing) tasks, which are dynamically created at run-time. Finalizing, related work, conclusions and future directions are exposed.

2. Handling Timing Concerns Using Aspects. In order to achieve dynamic rescheduling to improve tasks load-balancing, we investigate the use of aspect-oriented paradigms to cope with the modern system's crosscutting concerns, which are usually related to Non-Functional Requirements (NFR). Such requirements must be effectively handled already from requirements analysis to implementation phases to enhance system understandability during design. The context addressed by this work is similar to the one presented in the design of Distributed Real-time Embedded (DRE) systems, i. e. performance and timing NFR are very important during all application development phases. In this sense, we have adopted the taxonomy published in [6].

Traditional approaches, such as object-orientation, do not provide adequate means to deal with NFR handling. It occurs due to the inefficient modularization for NFR handling elements (timing requirements probes, serialization mechanisms, task migration mechanisms, among others), i. e. they are not modularized in a single or few system elements, but spread allover the system. Any change in one of these elements requires changes in different parts of the system, leading to a tedious and error-prone task that does not scale in the development of large and complex applications. The observation of these drawbacks motivates the use of an aspect-oriented approach, which makes possible to address such concerns in a modularized way. It separates the handling of the non-functional concerns in specific elements, increasing the system modularity, diminishing the coupling among elements, and though affecting positively the system maintainability, reuse and evolution [19]. Moreover specifically, the advantages of an aspect-oriented approach became clearer when applied to the task allocation strategies using heterogeneous platforms due to the need of profiling each task in different hardware, affecting several elements of the application. The use of aspects to address this concern represents an improvement since it helps to cope with the complexity in managing this concern spread through the whole system.

The next subsection presents a brief description of the NFR taxonomy presented in [6]. Following, a brief description of some aspects from an aspect framework, called DERAF [7], are presented to demonstrate how to deal with time-related NFR. Afterwards, an extension to DERAF, in terms of new aspects to deal with dynamic reconfigurable load-balancing, are is presented.

2.1. Non-functional Requirements. DRE systems domain presents a large set of NFR. Depending on the application domain, some requirements are more important than others. The same can be said about NFR handling: some of NFR are mandatory handled, while others are not. In this sense, Fig. 2.1 shows NFR taxonomy presented in [6], which focus on some of these very important requirements of DRE systems domain.

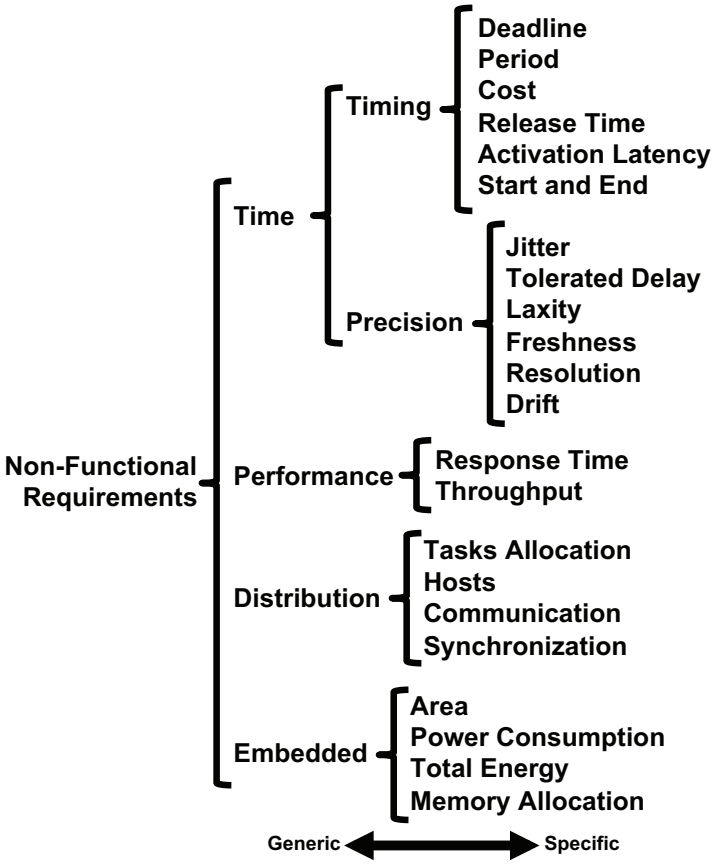


FIG. 2.1. *NFR requirements for DRE systems.*

The real-time concerns are captured by the requirements within the *Time* classification, which is divided in *Timing* and *Precision* requirements. The former presents time-related characteristics of system's tasks, activities, and/or action, e.g. deadlines or periodic executions. The later denotes constraints that affect the temporal behavior of the system in a "fine-grained" way, determining whether a system has hard or soft time constraints. An example is the *Freshness* requirement, which denotes the time interval within which a value of a sampled data is considered updated. Another key requirement is the *Jitter*, which directly affects system predictability since large variance in timing characteristics affects system determinism.

Performance requirements are not only tightly related to those presented in the *Time* classification, but also to those concentrated in the *Distribution* classification. They usually represent requirements employed to express a global need of performance, like the end-to-end response time for a certain activity performed by the system, or the required throughput rate in term of sending/receiving messages.

Distribution classification presents requirements related to the distribution of DRE system's activities, which usually execute concurrently. For instance, these concerns address problems such as task allocation over different PUs, as well as the synchronization and communication needs and constraints. Concerns related to embedded systems generally present requirements related to memory usage, energy consumption, and required hardware area size. *Embedded* classification gathers these concerns.

In this paper, the interest is to provide a runtime reconfigurable solution aiming at meeting time-related requirements. All mechanisms related to tasks migration among different PUs are non-functional crosscutting concerns, which are tightly related to system reconfiguration. In this sense, tasks migration is not only an expected final behavior of any system, but it also affects several functional elements in different ways and in different parts of the system.

2.2. Time-related Aspects. In order to address the mentioned *Time* and *Precision* requirements, this work (re)uses aspects from the *Distributed Embedded Real-time Aspects Framework* (DERAF) [7]. DERAF's Timing and Precision packages are presented in Fig. 2.2. A short description of each aspect is provided in the following paragraphs. Interested readers are pointed also to [7] for more details about DERAF.

TimingAttributes: adds timing attributes to active objects (e.g. deadline, priority, WCET, start/end time, among others), and also the corresponding behavior to initialize these attributes.

PeriodicTiming: controls execution of active objects by means of a periodic activation mechanism. This improvement requires the addition of an attribute representing the activation period and a way to control the execution frequency according to this period.

SchedulingSupport: inserts a scheduling mechanism to control the execution of active objects. Additionally, this aspect handles the inclusion of active objects into the scheduling list, as well as the execution of the feasibility test to verify if the active objects list is schedulable.

TimeBoundedActivity: limits the execution of an activity in terms of a deadline for finishing this activity, i. e. it adds a mechanism to restrict the maximum execution time for an activity, e.g. it limits the time which a shared resource can be locked by an active task. Jitter: measures the variance of activities' timing characteristics by means of measuring their start/end time, and calculating the variation of these metrics. If the tolerated variance was overran, corrective actions can be performed.

ToleratedDelay: restricts maximum latency for the beginning of an activity execution, e.g. limits the maximal duration in which a task can wait to acquire a lock on a shared resource.

DataFreshness: controls system data's expiration by means of associating timestamps to them, and also by verifying data validity before using them. Every time controlled data are written, their associated timestamps must be updated. Similarly, before reading these data, their timestamps must be checked [2].

ClockDrift: controls deviation of clock references in different PUs. It measures the time at which an activity starts, comparing it with the expected time for the beginning of this activity; it also checks if the accumulated difference among successive checks exceeds the maximum tolerated clock drift. If this is the case, some corrective action is performed.

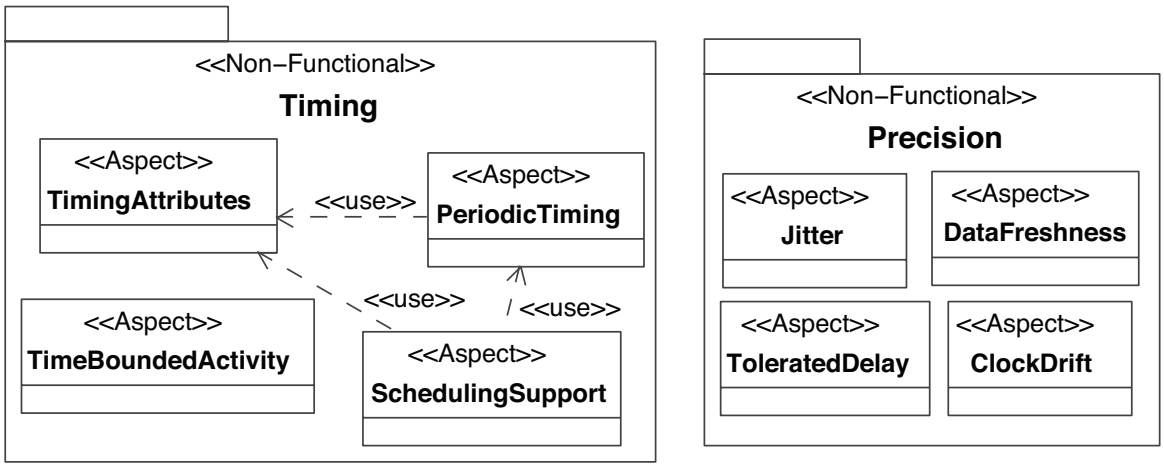


FIG. 2.2. *Timing and Precision packages from DERAF.*

2.3. Aspects to Support Tasks Self-Rescheduling. As mentioned before, task migration support characterizes a non-functional crosscutting concern in dynamic tasks rescheduling, spreading its handling mechanisms over several system's elements in a non-standard way. Therefore, we propose to use aspects to deal with this concern, and hence, two new aspects have been incorporated in DERAF: $TimingVerifier$, $TaskAllocationSolver$.

TimingVerifier and *TaskAllocationSolver* aspects use time parameters inserted by Timing package's aspects, and also services provided by aspects from the Precision package. To keep DERAF logical organization, both aspects have been included in an additional package, named TaskAllocation package, is included. Fig. 2.3 depicts the rescheduling-related package.

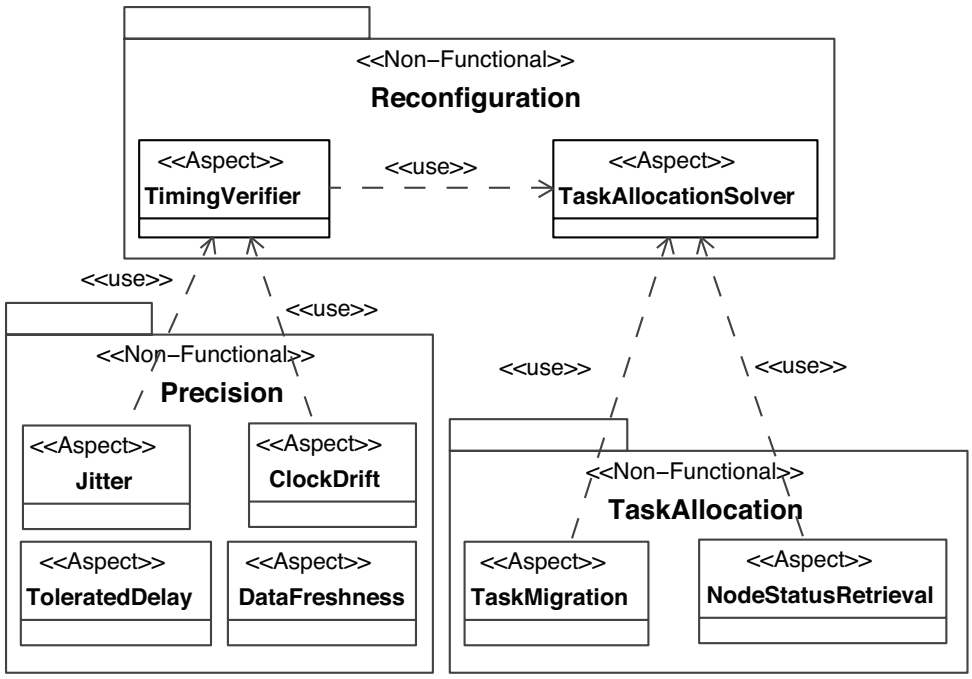


FIG. 2.3. *Aspects for Reconfiguration included in DERAF.*

TimingVerifier aspect is responsible for checking if PUs are being able to fulfill with timing requirements specified by *TimingAttributes*, *PeriodicTiming*, *ToleratedDelay* and *TimeBoundedActivity* aspects. In addition, *TimingVerifier* uses services provided by *Jitter* and *ClockDrift*.

To perform this checking, a mechanism, which controls if timing attributes are being respected, is inserted in the beginning and in the end of each task. More specifically, this mechanism consists in measuring the current time, comparing it with requirements specified by the correspondent attributes. For example, tasks deadlines' accomplishment can be checked by measuring the time in which a task actually finishes its computation, comparing this value with the time in which this task was supposed to finish. *TimingVerifier* uses the service of the *Jitter* aspects to gather information about the jitter related to analyzed requirement (in the mentioned example, the task's deadline). Moreover, considering the deadline again as example, *TimingVerifier* checks if the non-accomplishment of a task's deadline is constant, or if it varies in different executions or in the changing the platform scenario. In this sense, *TimingVerifier* can be used as base information, for instance, to know if the interaction among task is the responsible for the variance in tasks execution time.

ClockDrift aspect is used by *TimingVerifier* to gather information about synchronization among the different PUs, which is used, in addition to the time spent for task migration between PUs, to calculate the overall migration cost. To illustrate this idea, let's consider a task that has been migrated from a PU "A" to a PU "B", which is faster than PU "A" and potentially more capable of executing this task. The difference in the clock reference between these PUs could lead to an additional delay for this task's outcome (coming from PU "B") that would not be worth in comparison with letting the task to run in the PU "A".

The second key aspect for tasks dynamic rescheduling is the *TaskAllocationSolver*. It is responsible for deciding if a task will be migrated or not, and also for selecting to which PU this task is migrated. For that, *TaskAllocationSolver* checks the overload status of all destination PUs and the time spend for task migration, in order to decide if it is worthwhile to perform the migration. Hence, *TaskAllocationSolver* uses the measurements provided by *TimingVerifier* aspect. Based on these data, the reasoning about task reconfiguration feasibility is performed, as explained in the next section.

The reconfiguration itself and the retrieval of PUs status are performed by two other aspects from DERAF: *TaskMigration* and *NodeStatusRetrieval*. This way, reasoning and execution of tasks reconfiguration are decoupled, allowing that changes performed by one aspect do not affect the other one. A brief summary of the *TaskMigration* and *NodeStatusRetrieval* aspects is provided in the following.

TaskMigration: provides a mechanism to migrate active objects (tasks) from one PU to another one. It was originally used by aspects that control embedded concerns and, in the present work, is extended to provide services needed by *TaskAllocationSolver* aspect.

NodeStatusRetrieval: inserts a mechanism to retrieve information about processing load, send/receive messages rate, and/or the PU availability (i. e. "I'm alive" message). Before/after every execution of affected active objects (tasks), the processing load is calculated. Before/after every sent/received message, the message rate is computed. Additionally, PU availability message is sent at every "n" message or periodically with an interval of "n" time units. All of these information are taken into account by *TaskAllocationSolver* during the tasks migration decision process.

3. Dynamic Tasks Self-Scheduling. A task-based approach is then used, in which each task is designed to be an independent algorithm. They are grouped according derivation of the same high-level class, simplifying the managing of possible dependencies. Besides, it is coherent to assume that a group of tasks will have similar characteristics and hence would be desirable to execute in the same PU. However, this can lead to a non-optimal execution performance and must also be considered in the dynamic strategy discussed in the next sub-sections.

3.1. First Assignment of Tasks. For the first assignment of tasks, we do not use the modeled aspects, since tasks' real time measurements are unknown on first execution. One possibility is to perform the first schedule as a common assignment problem using Integer Linear Programming (ILP) and application timing requirements, similar to the approach used by [9]. This way, a set of tasks $i = 1$ to n have an implementation x and an execution cost estimation c on each PU j; and the allocation was following designed: the task i is not allocated on the processor j when $x_{i,j} = 0$ and the task i is allocated on the processor j when the $x_{i,j} = 1$. The constraints for the model were the maximum workload for the PUs. Bellow, the constraint of each processing unit j (U max), based on [9]:

$$U_j = \sum_{i=1}^{n} x_{i,j} c_{i,j} \leq U_{j_{\max}} \tag{3.1}$$

The best allocation was, then, found using the objective function that minimizes the resource utilization (percentage of occupancy for the PUs), defined as:

$$\left\{ \sum_{j=1}^{n} \sum_{i=1}^{n} x_{i,j} c_{i,j} \right\} \tag{3.2}$$

being the assignment variables $x_{i,j}$ the solution for the modeled ILP, m the number of computing units and n the number of considered tasks.

The mentioned ILP problem is of NP-hard complexity and become more complex in the scope of this work when dealing with more than two computing units and several tasks. To optimize the assignment calculation, some approaches concentrate on heuristics, as presented on [13].

However, this direction of estimating costs neither considers real execution times nor could represent the best assignment since a great number of estimations is used. This way, a second step of assignment allows taking into account real execution measurements extracted from the processors as well as dealing with the constraints presented by the NFRs. Based on that, the following dynamic module deals with real performance execution variables and possibly leads to a further better task assignment.

3.2. Task Scheduling Reconfiguration. After the first assignment, information provided by the profiling aspects is consideredconsidered. Based on involved estimated costs (previously calculated using the preprocessing approach of the first guess) and possible "interferences" of runtime conditions and new loaded tasks, one task can be rescheduled to run in other processing unit just if the estimated time to be executed in the new hardware will be less than the time in the actual unit, i. e., just if there is a gain. Simply, this relationship can be modeled in terms of the costs:

$$T_{reconfigPUnew} < T_{remainingPUold} - T_{estimatedPUnew} - T_{overhead} \tag{3.3}$$

where the remaining time ($T_{remainingPUold}$) and the estimated time ($T_{estimatedPUnew}$) are calculated, respectively, for the current PU and for the candidate unit based on previous measurements (or on the first assignment in the case of first rescheduling invocation). An overhead ($T_{overhead}$) is considered to calculate the execution time of the reconfiguration itself. The relationship between $T_{remainingPUold}$ and $T_{estimatedPUnew}$ is, then, the partial gain.

The information to calculate the rescheduling will be then provided by the TimingVerifier aspect and can be modeled as:

$$\begin{aligned} T_{reconfigPUnew} &= T_{setupReconfigPUnew} + T_{temporaryStorage} + \\ & \quad T_{transferRate} + T_{executionPUnew} + L \end{aligned} \tag{3.4}$$

where $T_{setupReconfigPUnew}$ represents the time for setting up a new configuration on the new processor; $T_{temporaryStorage}$ is the time spent to save temporal data (considering shared and global memory access); $T_{transferRate}$ measures the cost for sending/receiving data from/to the CPU to/from the new processing unit, which can be a bottleneck on the whole calculation; $T_{executionPUnew}$ symbolizes the measured or estimated cost of the task processed in the new unit; and **L** denotes a constant to represent possible system latency.

Reinforcing the concepts, this approach deals with runtime conditions, like input emphdata type and amount to be processed, tasks assignment, and instantiation of new tasks "on the fly". All these runtime parameters that could not be known a priori can influence the execution of the system and must be evaluated periodically, leading to a large number of reconfiguration analysis and decisions. Then, supposing that a determined task is going to be executed n times in a determined time window, the strategy bellow reschedules the formed queue of task instantiations, giving a new relation of gain, just if the following assumption occurs:

$$T_{reconfigPUnew} < \sum_{i=1}^{n} \left(T_{taskPUold_i} + T_{transferPUold_i} - T_{taskPUnew_i} - T_{transferPUnew_i} \right) \tag{3.5}$$

where **TreconfigPUnew** is the time to perform the reconfiguration (mainly data transfer from the current processing unit to the new one if the task needs the calculated data done until the time of rescheduling), **TtaskPUold** is the time performance of the task in the current unit, **TtransferPUold** is the time for transferring data from CPU to the current computing unit (via bus), **TtaskPUnew** is the assumed time performance of the task in the candidate processing unit, and **TtransferPUnew** is the time for transferring data from CPU to the candidate unit (via bus).

Algorithm 1, bellow, describes the task reallocation module. It is also important to mention that the heuristic needs improvements along future works.

Algorithm 1 Task Reallocation Heuristic.

1: Acquire Timing Data (Performance) about Previous Tasks Execution, storing them in a performance Database (Initialized according to First Assignment Phase and Regularly Updated);
2: Acquire information about PUs;
3: **if** Task has never been not executed **then**
4: Allocate it to a PUs according to First Assignment, storing it on the Database;
5: **end if**
6: Calculate Equation 3.5 according to Performance Data;
7: Execute Load-Balancing Algorithm;
8: Perform Reconfiguration Decision;
9: Reschedule Task to perform the Reconfiguration when applicable;
10: Store Performance Data in the Database;

4. Case Study: UAV-based Area Surveillance System. The use of the presented ideas is illustrated by a case study that consists of a fleet of Unmanned Aerial Vehicles (UAVs) in the context of area surveillance missions. This kind of system has several kinds of applications, such as military surveillance, borderline patrolling, and civilian rescue support in cases of natural disasters, among others. Fig. 4.1 illustrates a military surveillance usage scenario where the UAVs can also communicate with each other.

Such UAVs can be equipped with different kinds of sensors that can be applied, depending on the weather conditions, time of the day and goals of the surveillance mission [16]. In this case study, it is considered a fleet

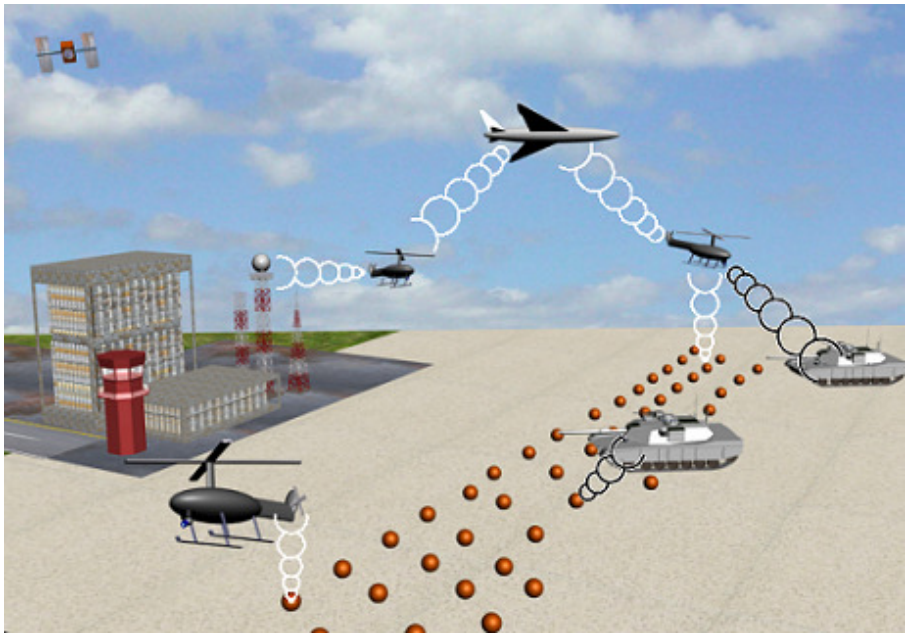


FIG. 4.1. *UAV-based Area Surveillance System.*

of UAVs that might accomplish missions during all the day and under whatever weather condition. UAVs must be able to provide different levels of information definition and detail, depending on the required data.

The UAVs receive a mission to survey a certain area, providing required data according to mission directions. Their movements are coordinated with the other UAVs in the fleet to avoid collisions and also to provide optimum coverage of the target area.

Each UAV is composed by six subsystems, making it capable to accomplish its mission and alto to coordinate with the others UAVs. These subsystems are: Collision Avoidance, Movement Control, Communication, Navigation, Image Processing, and Mission Management.

At this point, it is important to highlight the trade-off regarding cost, weight and size, and effectiveness of each UAV. The device, as a whole, may not be too big nor too heavy, in order to avoid unnecessary fuel consumption, as well as to be less susceptible of detection by counter forces sensors. Additionally, it may not have an enormous cost that could forbids the project. However, the UAV must be effective enough to provide the required data within an affordable cost and time budget. For more details about this trade-off discussion we address the readers to [16].

Another UAV's interesting feature is the possibility to apply different policies to missions, depending on user final intentions and specific requirements. There are two extremes for these policies: (i) Device Preservation Anyhow and (ii) Mission Accomplishment Anyhow. The first one consists of preserving UAVs even if the mission is not accomplished. It is especially applied in cases in which the devices can be destroyed and the information gathered by it is not worth compared to the cost of its destruction. On the other hand, in Mission Accomplishment Anyhow policy, the information gathered by the UAVs (and transmitted to the base station) is highly critical and overcomes the value of device loss. Within these policies, there are a variety of other factors that imposes different constraints to mission accomplishment and device preservation. Depending on the mission policy adopted, more resources can be (re)directed to tasks related to the movement control (when the UAV is escaping from a dangerous situation) or data gathering and processing (when information gathering has the highest priority).

In order to run the tasks described above, meeting the highlighted requirements and constrains modeled on the previous sections, we consider UAVs equipped with the following sensors: Visible Light Camera (VLC); SAR Radar (SARR) and Infra-Red Camera (IRC). To support the movement control and devices communication, each UAV is equipped with a hybrid "desktop"-based target platform which is used according to specific needs during the accomplishment of a certain mission, as detailed on section 4.2.

In this sense, each mentioned subsystem has a number of tasks to perform specialized activities related to a specific functionality, as depicted in the use cases diagram presented in Fig. 4.2. Based on the analyses of the

UAV functionalities, a summary of these tasks is provided in the following paragraphs.

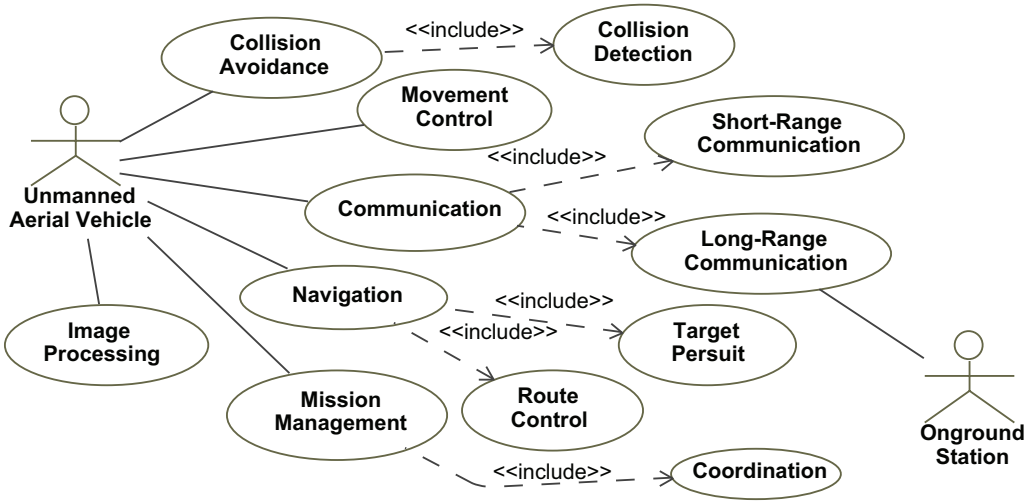


Fig. 4.2. *UAV Use Cases Diagram.*

Navigation guides the UAV movements, sending control information to the Movement Control subsystem. It is composed by the RouteControl and TargetPersuit tasks. The first task makes the necessary computation to guide the UAV through established waypoints, while the second one performs the same, but for dynamic waypoints that can be modified according to a moving object.

Image Processing gathers analog images, digitalizing them for further processing. It is composed by five tasks: (i) CameraController, which is responsible for camera movement, zoom and focus control of IRC and VLC, and antenna direction of SARR; (ii) Coder, which codifies the analog input into digital data; (iii) Compressor, which compresses the digital images; (iv) Reflectificator, which is responsible for the reflection in X and Y axis of radar image, as well as the rectification, that are necessary to avoid distortions in gathered images; (v) Filter, which filters radar images to eliminate the noise due to speckle effect [14].

Communication it has two main tasks: LongRangeCom and ShortRangeCom. The first task provides connectivity with pair communication nodes in long distances (of the order of kilometers), while the second one provides connectivity in short range distances (of the order of meters). These two tasks uses a third one, called Codec, which code and decode data transmissions.

Mission Management has also two tasks: MissionManager and Coordinator. The first one manages the information about the mission, such as required data, mission policy and resource autonomy control (e.g. remaining fuel). On the other hand, the second one drives the coordination with the other UAVs to avoid overlapping in the surveillance area.

Collision Avoidance is composed by two tasks: CollisionDetector, which detects possible collisions with other UAVs of the fleet or non cooperative flying objects; and CollisionAvoider, which calculates UAV's collision escape directions, sending them to the Movement Control subsystem.

4.1. Execution Platform. The target architecture of each UAV is composed of a four heterogeneous PUs platform: one host (the CPU), two GPUs, a PPU, and a PCICC. Fig. 4.3 shows the desired platform, where the *Profiling* gathers information from PUs (tasks performance) and the *Reconfiguration* distributes the tasks along them (intra allocation) according to the presented algorithms. It also consider sending data to be processed by other UAVs (inter allocation).

4.2. Reconfiguration Approach. Starting the mission, the UAVs have an initial task allocation throughout the CPU and the PU devices according to sub-section 3.1. In the current experimentation, it was considered to use the ILP approach for the first distribution using the GLPK toolkit [8]. Table 1 exhibits estimated costs (based on [3]) and first tasks' priorities that feed the GLPK-based simulation of task scheduling.

During execution, the mechanisms injected by TimingVerifier and the aspects Jitter and ClockDrift will start to generate information related to timing measurements. The TimingVerifier aspect will provide data to TaskAllocationSolver, which will get data from NodeStatusRetrieval. With the reasoning mechanisms, it will periodically analyze the provided information according to the algorithm introduced in 3.2.

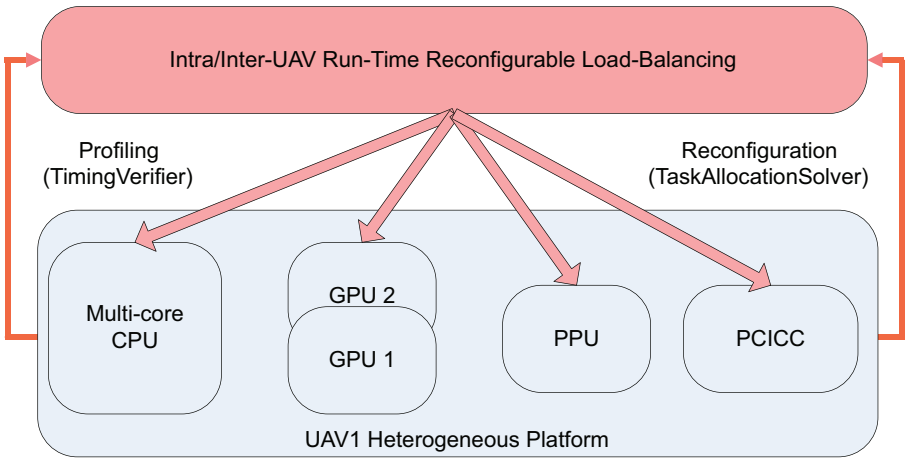


FIG. 4.3. *Execution platform.*

TABLE 4.1
Task Estimation Costs.

Task	**Estimated Cost (scale: 1 to 6)**				**First Priority (scale: 1 to 6)**
	GPU	**PPU**	**CPU**	**PCICC**	
Image Processing	1	4	6	—	1
Collision Avoidance	3	2	5	—	2
Movement Control	2	2	3	—	1
Navigation	1	1	2	—	3
Communication Short/Long range	4/6	—	3/5	1/2	4
Mission Management	5	—	1	—	6

Emphasis is given to the RIP subsystem, which is considered to be the group that requires more processing due to the handling of large data and new instantiations created dynamically. The RIP workflow is depicted on Fig. 4.4. Theoretically, tasks associated to RIP should execute with better performance on a GPU device when the application is not aware about the context of the whole execution scenario, i. e. executing stand alone. Thus, initial demands of all tasks should be executed in the GPU. Shortly, the captured data (raw scalar image) must be "adjusted" regarding the SAR position parameters (range and azimuth), followed by Fast Fourier Transform (FFT), image rotation, and other corrections to produce the final image. This process can be performed individually in the range and azimuth directions and it consists basically in a data compression on both directions using filters that maximize the relation between the signal and the noisy. Readers are addressed to [5] to get refined explanations about the workflow.

Afterwards, in the explored surveillance system, the final image is submitted to a pos-processing in order to identify regions of interest that could contain objects specified in the mission directions as a "pattern to be found" or a "target". In this case, more resolution on specific areas will be needed and new data will be generated, demanding more processing from the assigned PU(s) in order to produce new images and extract relevant information (patterns).

Based on that description, this dynamic scenario clearly influences the tasks' priority since, at a moment, the new high-resolution images will have higher priorities if compared to others that became more "generic". These events cannot be predicted a priory and the verification of such situation require a smart, context-aware, and dynamic reconfiguration support to balance the workload, accomplishing the timing requirements and budget.

4.3. Results. Considering 2 UAVs in the case study, Table 4.2 denotes the behavior of the dynamic reschedule load-balancer simulator. The "first guess" represents one instantiation of each group of tasks assigned to a PU; and with dynamic creation of new groups (4, 8, and 12) of RIP tasks, the assignment is changed and optimized to minimize the total execution time. Note that these values cannot represent the best assignment since the version of the simulator did not consider all parameters that influence the whole system. As it is an

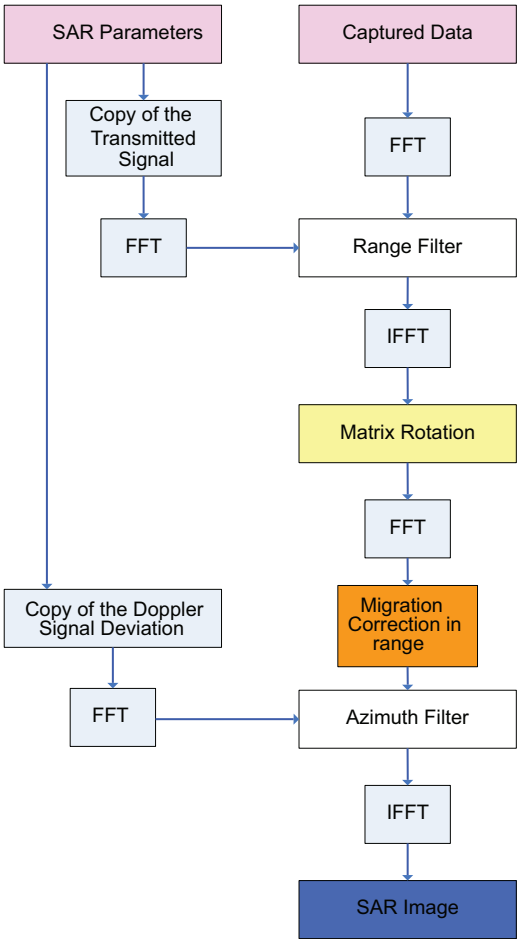


FIG. 4.4. *SAR Image Processing (based on the notes of [5]).*

TABLE 4.2
Task Assignment.

Task	**1st Gess**	**Dynamic Image Processing Created Tasks**		
		4	**8**	**12**
Image Processing	GPU1	GPU1 GPU2	GPU1 GPU2 PPU	GPU1 GPU2 UAV2-GPU1
Collision Avoidance	GPU2	PPU	CPU	CPU
Movement Control	PPU	PPU	CPU	CPU
Navigation	PPU	PPU	PPU	CPU
Communication	CPU	CPU	CPU	PCICC
Mission Management	CPU	CPU	CPU	CPU

ongoing work, more accurate data about the reschedule must be provided along the simulator's refinement in order to represent the scenario as realistic as possible.

5. Related Work. VEST (Virginia Embedded System Toolkit) [15] is a set of tools that uses aspects to compose a distributed embedded system based on a component library. Those aspects check the possibility of composing components with the information taken from system models. It provides analysis such as task schedule feasibility. However, it performs statically analysis at compiling time. In our proposal, aspects are used dynamically to change the system configuration at runtime, adapting its behavior to new operating conditions.

Although there are some related works concerning dynamic reconfiguration in cluster computing, like for example ([18]; [12]; [1]), our approach concentrates on single desktop platforms composed by different processing units where the reconfiguration is performed within these devices. In this way, the work presented by [9] implements dynamic reconfiguration methods for Real-Time Operating System services running on a Reconfigurable System-on-Chip platform based on CPU and FPGA. The method, based on heuristics, take into account the idleness percentage of the computing units and unused FPGA area (calculated as pre-processing) to perform the load-balancing and to decide about a reconfiguration of tasks in runtime by means of task migration. Our approach complements this related work, developing generically methods that comprise more than two processing units and that work with dynamic performance data.

Targeting GPUs, the work of [17] presented a programming framework to achieve energy-aware computing. On the proposed strategy, the compiler translates the framework code to a C++ code for CPU and a CUDA code for GPU. Then, a runtime module dynamically selects the appropriate processor to run the code taking into account the difference in energy efficiency between CPU and GPU based on energy consumption estimation models. However, it does not take into account runtime energy measurements (runtime profiling), which is an important module of our work.

Another approach, focusing performance improvement of spheres collision detection simulation, was proposed by [10], in which some strategies have been presented to perform data balancing over CPU and GPU, both in an automatically and manually options. That work takes into account the performance of a kernel implemented on the CPU and GPU. After the execution starts, both versions of the programs are executed with equally input data and time performance is verified. More data are then dynamically assigned to the processor that executed faster the previous data, indicating that the approach uses data decomposition instead of task decomposition. Our work concentrates on task decomposition and its dynamic assignment according to estimated or profiled performance.

The work presented in [4] published a study to accelerate compute-intensive applications using GPUs and FPGAs, listing some of their pros and cons. The work performed a qualitative comparison of application behavior on both computing units taking into account hardware features, application performance, code complexity, and overhead. Although GPUs can offer a considerable performance gain for certain application, that work's results showed that FPGAs can be an interesting computing unit and could promote a higher performance compared to GPU when applications require flexibility to deal with large input data sets. However, using FPGAs comes with cost of hardware configuration before using it as a computing unit, a task usually oriented to experienced users. Thus, task reconfiguration frameworks, as the one presented in this work, could provide a higher abstraction layer to assist developers during system design.

6. Conclusions and Future Work. This paper presents a methodology to address the problem of efficient task assignment in runtime targeting hybrid computing platforms. It allows the use of resources offered by an asymmetric computer platform, providing compliance with dynamic changes in timing requirements and constraints, and also runtime conditions. In order to achieve the proposed goals, our proposal uses an aspect-oriented framework in conjunction with a dynamic task self-rescheduling strategy, in order to address the dynamic runtime scenarios under concern.

A UAV-based surveillance system simulation has been used to show the need for workload adaptation required by sophisticated applications, running on top of hybrid computers, which face dynamic execution scenarios. Real-time task rescheduling was applied on UAV PUs, focusing on RIP. Results indicate that rescheduling contributes to a more appropriate system resource usage, and hence towards performance improvement. Sending/receiving data between UAVs was also considered, but details about specific problems related to these interactions, such as delays in the communication between the UAVs, have not been focused by this text.

Future directions lead to refine the scheduling strategy to provide complete simulations, considering a larger range of runtime parameters, including the reconfiguration costs itself; and real algorithms for UAV's subsystems, emphasizing RIP dynamicity. Heuristics to predict the future allocation of tasks based on its recent use seems to be a good strategy, and will possibly avoid unnecessary reconfigurations in a specific time-window.

Acknowledgments. A. P. D. Binotto thanks the partial support given by DAAD fellowship and the Programme Alßan, the European Union Programme of High Level Scholarships for Latin America, scholarship no. E07D402961BR.

E. P. Freitas thanks the Brazilian Army for the given grant to pursue the PhD program focused on Embedded Real-time Systems at Halmstad University in cooperation with UFRGS (Brazil).

REFERENCES

[1] B. B. Brandenburg, J. M. Calandrino, and J. H. Anderson, *On the Scalability of Real-time Scheduling Algorithms on Multicore Platforms: a case study*, Proceedigns of Real-Time Systems Symposium, (2008), pp. 157–169.

[2] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, F. Di Gi, B. Omenico, K. Ramamritham, J. Stankovic, and L. Strigini, *The Meaning and Role of Value in Scheduling Flexible Real-Time Systems*, Proceedigns of Early Aspects: Current Challenges and Future Directions, 46 4 (2000), pp. 305–325.

[3] G. Cai, K. Peng, B. M. Chen, and T. H. Lee, *Design and Assembling of a UAV Helicopter System*, Proceedigns of International Conference on Control and Automation, (2005), pp. 697–702.

[4] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, *Accelerating Compute-Intensive Applications with GPUs and FPGAs*, Proceedigns of Symposium on Application Specific Processors, (2008), pp. 101–107.

[5] I. Cumming and F. Wong, *Digital Processing of Synthetic Aperture Radar Data*, Artech House-London, 2005.

[6] E. P. Freitas, M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva Jr., and F. C. Carvalho, *Using Aspect-Oriented Concepts in the Requirements Analysis of Distributed Real-Time Embedded Systems*, Proceedigns of International Embedded Systems Symposium, (2007), pp. 221–230.

[7] E. P. Freitas, M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva Jr., and F. C. Carvalho, *DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design*, Proceedigns of Early Aspects: Current Challenges and Future Directions, (2007), pp. 55–74.

[8] The GNU Project, *GLPK—GNU Linear Programming Kit*, In http://www.gnu.org/software/glpk/. Access in Jun. 2008.

[9] M. Götz, F. Dittmann, and T. Xie, *Dynamic Relocation of Hybrid Tasks: A Complete Design Flow*, Proceedigns of Reconfigurable Communication-centric SoCs, (2007), pp. 31–38.

[10] M. Joselli, M. Zamith, E. Clua, A. Montenegro, A. Conci, R. Leal-Toledo, L. Valente, B. Feijo, M. Dornelas, and C. Pozzer, *Automatic Dynamic Task Distribution between CPU and GPU for Real-Time Systems*, Proceedigns of the IEEE International Conference on Computational Science and Engineering, (2008), pp. 48–55.

[11] G. Kiczales, J. Irwin, J. Lamping, J. M. Loingtier, C. Videira Lopes, C. Maeda, and A. Mendhekar, *Aspect-Oriented Programming*, Proceedigns of European Conference for Object-Oriented Programming, (1997), pp. 220–242.

[12] M. Linderman, J. Collins, H. Wang, and T. Meng, *Merge: A Programming Model for Heterogeneous Multi-core Systems*, ACM Sigplan Notices, 43 3 (2008), pp. 287–296.

[13] M. McCool, *Scalable Programming Models for Massively Multicore Processors*, Proceedings of the IEEE, 96 5 (2008), pp. 816–831.

[14] M. I. Skolnik, *Introduction to Radar Systems*, Third ed., McGraw-Hill, 2001.

[15] J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis, *VEST: Aspect-Based Composition Tool for Real-Time System*, Proceedigns of Ninth IEEE Real-Time and Embedded Technology and Applications Symposium, (2003), pp. 58–69.

[16] D. M. Stuart, *Sensor Design for Unmanned Aerial Vehicles*, Proceedigns of IEEE Aerospace Conference, (1997), pp. 285–295.

[17] H. Takizawa, K. Sato, and H. Kobaiashi, *SPRAT: Runtime Processor Selection for Energy-aware Computing*, Proceedigns of the IEEE International Conference on Cluster Computing, (2008), pp. 386–393.

[18] P. Wang, J. Collins, G. Chinya, H. Jiang, X. Tian, M. Girkar, N. Yang, G. Y. Lueh, and H. Wang, *EXOCHI: Architecture and Programming Environment for a Heterogeneous Multi-core Multithreaded System*, Proceedigns of the ACM SIGPLAN conference on Programming language design and implementation, (2007), pp. 156–166.

[19] M. A. Wehrmeister, E. P. Freitas, D. Orfanus, C. E. Pereira, and F. Ramig, *A Case Study to Evaluate Pros/-Cons of Aspect- and Object-Oriented Paradigms to Model Distributed Embedded Real-Time Systems*, Proceedigns of 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, (2008), pp. 44–54.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 291–305. http://www.scpe.org
ISSN 1895-1767

WIRELESS SENSORS AND ACTUATORS NETWORKS: CHARACTERIZATION AND CASES STUDY FOR CONFINED SPACES HEALTHCARE AND CONTROL APPLICATIONS*

DIEGO MARTÍNEZ† AND FRANCISCO BLANES, JOSE SIMO, ALFONS CRESPO‡

Abstract. Nowadays developments in Wireless Sensor and Actuators Networks (WSAN) applications are determined by the fulfillment of constraints imposed by the application. For this reason, in this work a characterization of WSAN applications in health, environmental, agricultural and industrial sectors are presented. Two cases study are presented, in the first a system for detecting heart arrhythmias in non-critical patients during rehabilitation sessions in confined spaces is presented, as well as an architecture for the network and nodes in these applications is proposed; while the second case presents experimental and theoretical results of the effect produced by communication networks in a Networks Control System (NCS), specifically by the use of the Medium Access Control (MAC) algorithm implemented in IEEE 802.15.4.

Key words: real-time systems, wireless sensor and actuator networks, embedded systems, real-time monitoring and control

1. Introduction. Currently, there is a great interest in developing applications for monitoring, diagnosis and control in the medical, environmental, agricultural and industrial sectors, to improve social and environmental conditions of society, and increasing quality and productivity in industrial processes. The development of Wireless Sensor and Actuators Networks (WSAN) applications will contribute significantly to solve these problems, and facilitate the creation of new applications.

Some applications which can be developed using WSAN are:

- Medical Sector: economic and portable systems, to monitoring, recording and analyzing physiological variables, from which it is possible to indicate the status of a patient and detect the presence or risk of developing a disease. As well, developing systems for the detection and analysis of trends in the daily behavior of patients, contributing to timely detect the presence of a health problem, and providing an economically viable solution to patient care in societies where the old population is great.
- Environmental Sector: continuous systems monitoring of species in dangerous extinction, monitoring and detection of forest fire systems, etc.
- Agricultural sector: detection systems, microclimates monitoring and pest control, to reduce the use of agrochemicals and make an optimal control of pests; optimal use of water in irrigation systems, etc.
- Industry: economic systems and easy installation for monitoring, diagnosis and control of plants and industrial processes.

Some of the currently technological challenges in WSAN development are [1], [2], [3], [4], [5]:

- It is necessary to develop detailed models of the system components (hardware and software tasks, task scheduler, medium access control and routing protocols), in languages that allow correct specifications and the subsequent analysis of information processing, reachability, security, and minimum response time application, enabling analysis of end to end deadline in real time applications.
- Task scheduler and medium access control and routing protocols proposed in this area are mostly focused on the optimization of a single critical parameter of the application, which often affects considerably the performance of the others. Therefore it is necessary to create new cooperation forms between these levels of the application architecture, in order to take the most appropriate decisions for the system reconfiguration in relation to the application's quality of service (QoS). Additionally, current proposals consider QoS parameters directly linked to conventional parameters of the operation and communication between computers but not to particular application requirements, so do not allow achieving optimal performance in applications.
- It is necessary to develop analysis strategies for performance and stability of signal processing and control algorithms in this area, in order to guide the design towards a co-design methodology to develop the processing algorithm and the implementation of computer architecture, allowing to compensate the sampling period changes and jitter effects, and optimize other parameters such as power consumption.

*This work has been partially supported by the project D2ARS (CYTED)
†Departamento de Automática y Electrónica, Universidad Autónoma de Occidente. Cali, Colombia (dmartinez@uao.edu.co)
‡Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia. Valencia, España ({patricia, pblanes, jsimo, acrespo}@disca.upv.es)

The previous paragraphs show how these developments are determined by the fulfillment of constraints imposed by the application, such as energy consumption, limited computing power, coverage of large areas and real time deadlines, etc. For these reasons, in this work a characterization of WSAN applications for health, environmental, agricultural and industrial sectors is presented, then two cases study are presented; in the first a system for detecting heart arrhythmias in non-critical patients during rehabilitation sessions in confined spaces is presented, as well as an architecture for the network and nodes in these applications is proposed; while the second case presents experimental and theoretical results of the effect produced by communication networks to the Networks Control Systems (NCS), specifically by the use of the Medium Access Control (MAC) algorithm implemented in IEEE 802.15.4.

The article is organized as follows, section 2 shows a classification and characterization of applications in health, environmental, agricultural and industrial sectors, section 3 presents the case analyzed, in section 4 a proposal for the nodes architecture is presented, the network architecture and its simulation results are presented in section 5, finally in section 6 the conclusions and future work are presented.

2. Classification of Applications. During the classification was detected that different application sectors share similar characteristics from a technological point of view, for this reason the classification and characterization was developed in five types of application rather by sectors [6], [7], [8], [9], [10], [11], [12], [13], [14].

- Type 1 applications are characterized by measuring sampling periods from one second to few hours, and no strict deadlines for the generation of the algorithms results. Additionally, these applications, developed in open spaces, must cover large areas and it is necessary to synchronize measurements in different nodes. Agricultural and environmental applications, designated to measure, record and to analyze environmental variables, primarily belong to this category. The energy sensors autonomy expects for each node varies from days to months; in some applications, in places without access to conventional energy sources, nodes are equipped with energy transducers like solar cells, which supply energy to the nodes batteries.
- Type 2 applications are developed in confined spaces and have greater computing capacity demanding than applications type 1, although there is not strict response time, either. Because they are in confined spaces and nodes are fixed, there are no restrictions on energy consumption since they can use conventional energy sources; however, the use of wireless networks is justified since it facilitates the installation, adaptability and portability of implementation, in addition to the lower costs of implementation. Such applications are fined mainly in industrial and agricultural sectors. The sampling periods range from one millisecond, for implementation of diagnostic algorithms, until a second for monitoring and supervision tasks. The diagnostic algorithms do not require continuous operation; therefore the samples can be stored before being processed.
- Type 3 applications. In this category, in addition to measuring and processing data requirements similar to those in applications type 2, grouped applications are required to process images and they are developed in open spaces, some of which are in the agricultural sector for the detection of pests, and environmental sector aimed at detecting fires. Some of the nodes are mobile and require few hours' energy autonomy, and then restrictions in terms of power consumption are large. At the same time it is also necessary for synchronization of the nodes. While, because of the algorithms used for image processing, the computing capacity, memory size and communication bandwidth requirements are greater than applications type 2.
- Type 4 applications. These applications differ from applications type 3 because they are developed in confined spaces, then the network's coverage is not demanding; energy sensors autonomy expected are also higher, becoming close to one week. Grouped healthcare applications to the detection of diseases are in this category with body area networks (BAN).
- Type 5 applications. In these applications a sample data should be sent every sampling period, and then sampling periods are limited by the minimum interframes time space of data communication protocols. For this category, a range of sampling periods between 50 milliseconds and a few seconds has been selected. In this category are the applications of industrial control process, which are developed in confined spaces, so the distance between nodes is not big, and there are not restrictions on energy consumption. The deadlines for generating actions are less than or equal to a sampling period, and it is necessary to guarantee end to end deadline. If these constraints are not fulfilled, the control system performance can be degraded significantly, even generating instability in the system, therefore, it is

important to synchronize the activities of the nodes that are integrated in the control loop. In addition, to improve the control system performance, it is important to limit the variability in the task jitter.

Table 2.1 summarizes the characteristics of the applications described. As a strategy to increase reliability in the presence of faults, and optimizing the applications QoS, this proposal also has considered the migration of components between the nodes, which will be reflected in the architecture of the network and nodes.

TABLE 2.1
Analysis of requirements for each application type.

Application	Computing capacity	Memory size	Communication bandwidth	Location	Node mobility	Real-Time	Networks's coverage	Energy autonomy	Syncro-nization
Type 1	Low performance	Low	$< 256kbps$	Yes	No	Only measurement	Open space 10 km	Months	Yes
Type 2	Low performance	Medium	$< 256kbps$	Yes	No	Only measurement	Confine space 100 m	There isn't restriction	Yes
Type 3	High performance	High	$1Mbps$	Yes	Yes	Only measurement	Open space 10 km	Hours	Yes
Type 4	High performance	Medium	$< 256kbps$	Yes	Yes	Only measurement	Confine space 1 km	Days	Yes
Type 5	Low performance	Low	$< 256kbps$	No	No	End-to-end and minimum jitter variability	Confine space 100 m	There isn't restriction	Yes

3. Arrhythmia Detection Algorithm. Actually cardiovascular problems have the highest mortality rate from natural causes in the world. The great interest in developing devices for clinical detection and continuous monitoring of such diseases, is based on these activities are limited by the information type and the moment that it is caught, so transitional abnormalities can not be always monitored. However, many of the symptoms associated with cardiovascular diseases are related to transient episodes rather than continuing abnormalities, such as transient surges in blood pressure, arrhythmias, and so on. These abnormalities can not be predicted therefore a controlled supervision analysis is discarded. The reliable and timely detection of these episodes can improve the quality of life of patients and reduce the therapies cost. For this reason in this work a WSAN architecture to address such problems is proposed, this application belongs to type 4 described in paragraph 2.

As an example, the detection of arrhythmias using data from electrocardiogram (ECG) measure, in patients who are moving during a rehabilitation activity in a confined space of 100m x 100m, as a rehabilitation center, was analyzed. The sampling period was selected from the ECG frequency spectrum, which, according to the American Heart Association, has 100Hz harmonics. The greatest amount of relevant information for monitoring and detection of arrhythmias is between 0.5Hz and 50Hz.

When analyzing the ECG frequency spectrum can be established that the relevant components of the signal (QRS complex and waves P and T) are up to 35Hz. Applying the sampling theorem a minimum sampling period of 14ms approximately is necessary, but for practical purposes a period of 3ms was selected.

For the detection and analysis of ECG the Pan and Tompkins algorithm was selected, [15]. The results of this algorithm are used by a maximums detection algorithm, which identifies the time when segments of the ECG wave were presented, figure 3.1. Subsequently the analysis of the separation time between two R waves, the duration of the QRS segment and the energy of the wave R is developed, which allows detecting the presence of arrhythmias [16].

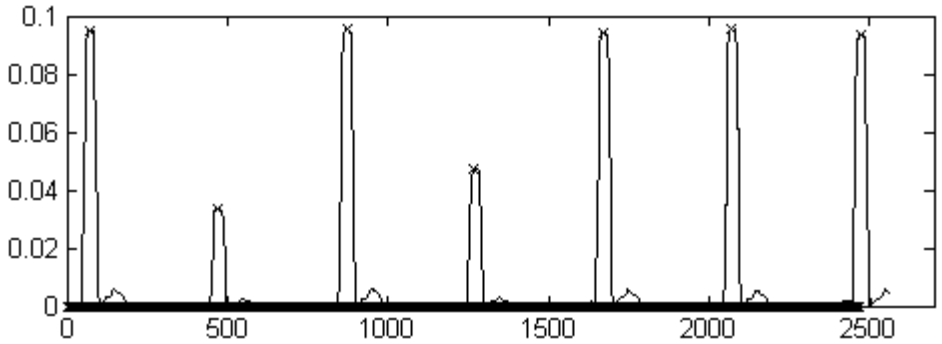


FIG. 3.1. *Results of maximums detection algorithm.*

4. Architecture Node. The proposed generic architecture for the network nodes in applications type 4 is presented in figure 4.1. Its characteristics are:

- The architecture enables the hardware and software components co-design. This feature will allow optimizing the development of distributed application components required for its implementation in hardware and software, getting a balance between cost, power consumption and processing time.
- There are fixed and mobile nodes. The latter are linked to the sub-network that guarantees them the best QoS (QoSsn) during its movement (less saturated sub-networks).
- Communication between the nodes and local coordinators is done through wireless networks.
- Use an EDF scheduler and dynamic scaling voltage and frequency techniques of the processor to optimize the power consumption [17]. This allows fulfilling the application deadlines, which is supported on statics utilization rate tables for each operating frequency, and periods of execution for each task.
- Update its QoS indexes (QoSn); using these indexes and its neighboring ones it is possible to request to another node in its sub-network the migration, creation or destruction of components (some of which are clones of others).

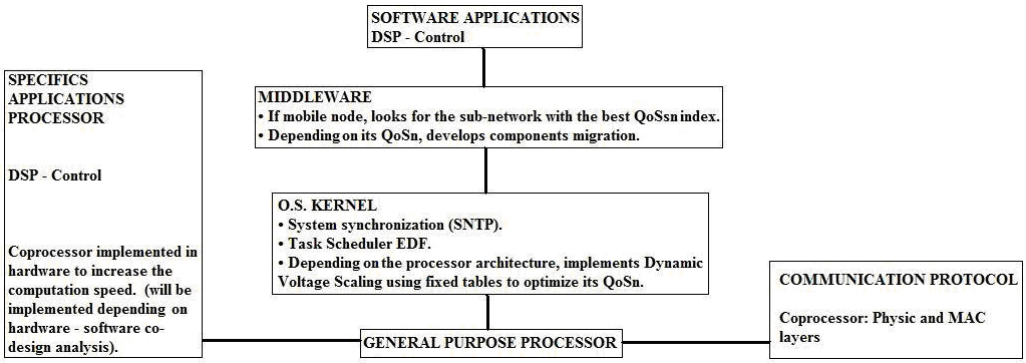


FIG. 4.1. *Nodes architecture.*

To select a set of architectures for an adequate performance of these applications, the performance of the arrhythmia detection algorithm, presented in section 3, was analyzed on four types of processors currently used to implement nodes in sensor networks: ARM7TDMI, MSP430, PIC18 and MC9S08GB60. For the analysis, the same operation velocity for each processor was used, 8MIPS. The time necessary to develop the Pan and Tompkins algorithm is presented in Table 4.1, which was estimated considering the sum of the values of individual functions (derivative, quadratic function and integrator window) in each architecture.

TABLE 4.1
Computing time to develop the Pan and Tompkins algorithm.

Processor	Derivative	Quadratic function	Integrator window	Total computing time	Period	Percentage of utilization (U)
LPC2124-ARM	$70.2\mu s$	$142\mu s$	$280.5\mu s$	$492.7\mu s$	$3000\mu s$	16.4%
MSP430F1611	$191.9\mu s$	$162.5\mu s$	$697.8\mu s$	$1052.2\mu s$	$3000\mu s$	35%
PIC18F458	$406.2\mu s$	$209\mu s$	$1083.7\mu s$	$1698.9\mu s$	$3000\mu s$	56.6%
MC9S08GB60	$497.2\mu s$	$332\mu s$	$707.35\mu s$	$1536.55\mu s$	$3000\mu s$	51.3%

The results show that the ARM architecture requires a lower percentage of utilization, while the PIC architecture needs the highest utilization percentage.

It also was related consumed power by each architecture in active mode (P_A) with the respective percentage of utilization during the implementation of the algorithm, table 4.2. It can be seen as the ARM7 architecture has a closer performance to the architecture MC9S08GB60; then these two architectures are appropriate for the implementation of the case proposed. The MSP430 architecture presented the best indicator.

TABLE 4.2
*Indicator PA*U.*

	Utilization percentage (U)	Active Power (P_A) [mW]	$P_A * U$
LPC2124-ARM	16.4%	180	29.52
MSP430F1611	35%	19.2	6.72
PIC18F458	56.6%	220	124.52
MC9S08GB60	51.3%	51.6	26.47

5. Network Architecture for Confined Spaces Healthcare Applications. In figure 5.1, a generic architecture for network applications type 4, which integrates different types of nodes, is proposed. The approach of a cooperation plan between architecture levels of the application, in order to take the most appropriate decisions for the reconfiguration of the system in relation to the application QoS, can be appreciated. General goals of the architecture are:

- Minimize latencies.
- Optimize power consumption.

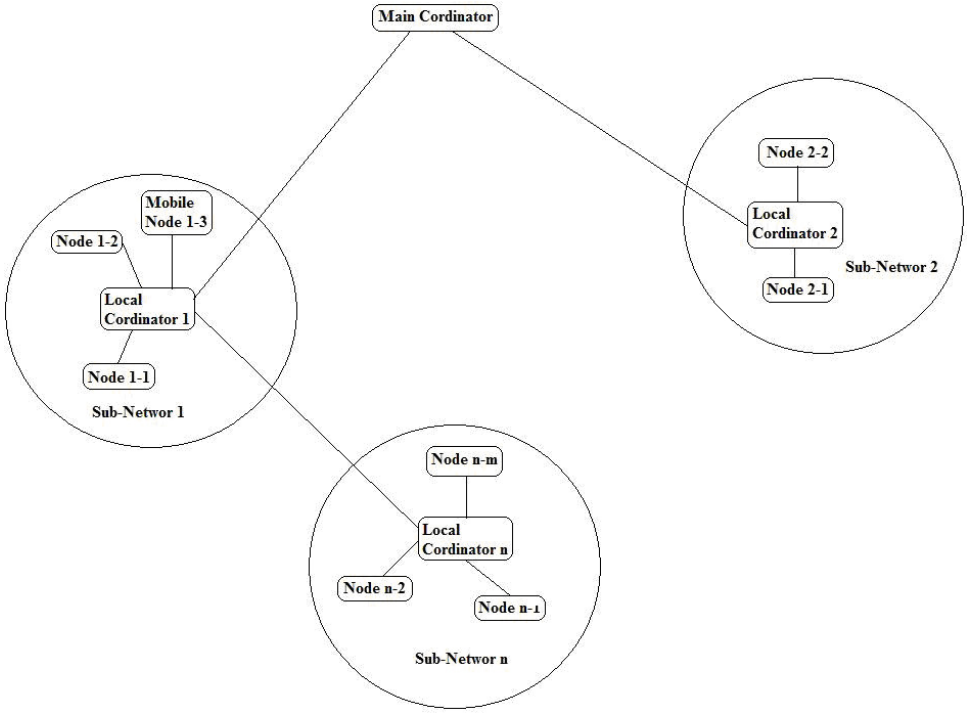


FIG. 5.1. *Network architecture.*

The Main Coordinator is responsible for coordinating the complete application. It will have a fixed location, and communication with local coordinators will be supported through wireless or wired links. It develops the following functions:

- Send synchronization signals to the local coordinators of the sub-networks.

Local Coordinator controls the activity inside the sub-network and develops some information processing activities, whose architecture is presented in figure 5.2 and its features are:

- It has a fixed location.
- Sends synchronization signals to nodes in its sub-network.
- Develops routing packets between sub-networks using multihop techniques.
- Distributes QoS indexes of nodes which belong to its sub-network (QoSn).
- Calculates its sub-network QoS index (QoSsn= f(quantity of information to be transmitted)), and distributes this value and its neighboring sub-networks indexes (those reached in a single communication hop) between nodes in its sub-network. Depending on which:
  - Accepts linking new nodes to sub-network.
  - Updates best routes in the routing tables of data (which will be function of hops and the utilization percentage—information transmitting—of each router node).

As a first approximation to the proposed architecture, we examined the performance of the case analyzed on the IEEE 802.15.4 protocol. Considerations for the proposed solution to the case are:

- Transmission of the analysis results, from nodes located on each patient to a main node, every 3 s. The data frame consists of 2 Bytes, which contain patient codes and the type of arrhythmia detected.
- After each sending the sender node waits for an acknowledgement (ACK) from next node in the routing path. If there isn't an answer before 100ms the node sends again the information. If after 25 attempts

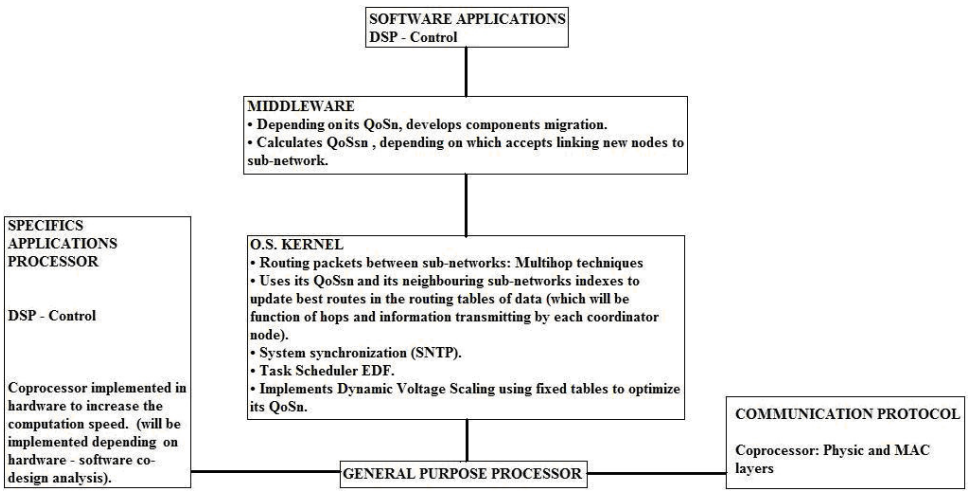


Fig. 5.2. *Sub-network coordinator architecture.*

there is no answer this node changes to a mistake state.

- The communication protocol selected is IEEE 802.15.4. The node distribution is shown in figure 5.3, which allows covering all possible locations of patients considering the specifications of the devices selected to implement the physical layer, CC2420, whose characteristics are:
  - Coverage radio of 30m, and 100m without obstacles.
  - Frequency range of 2.4–2.4835 GHz.
  - Supports data transfer rates of 250 kbps.

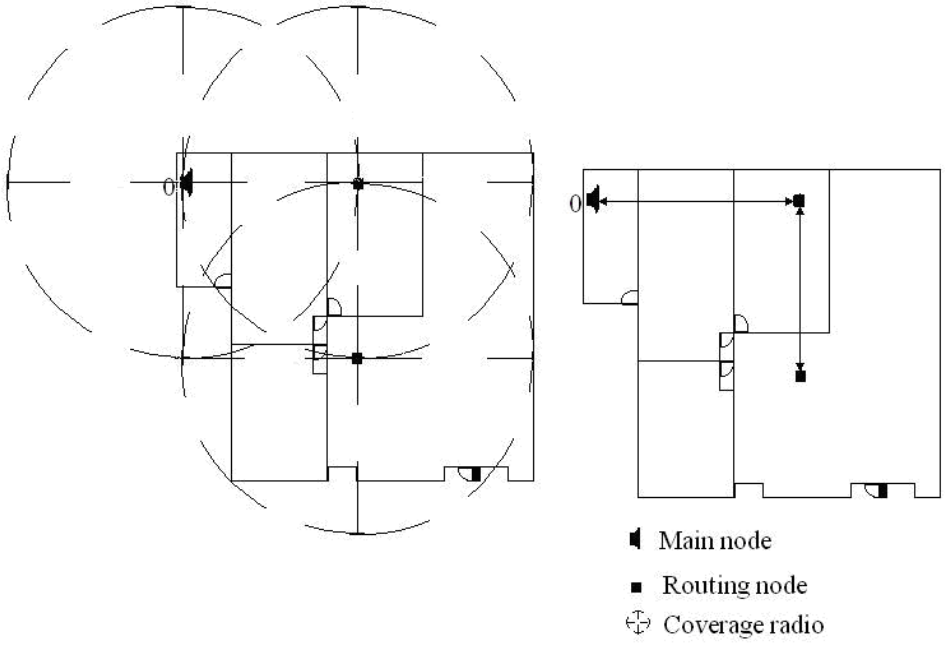


Fig. 5.3. *Distribution nodes for case and their coverage.*

In the case a network as presented in figure 5.4 was proposed. It consists of 3 fixed nodes which have no restrictions on power consumption, will receive reports from five patients and route the messages to the main node. The fixed devices have fixed identifiers 0, 1 and 2; the main node has the 0 identifier, and devices on every patient have identifiers from 3 to 7. The routing is developed through 1, 2 and 0 nodes, 0 is the network coordinator, each of these nodes forming a sub-network together with patients, figure 5.4. The mobile nodes leave and enter the sub-networks continuously changing the configuration and network structures.

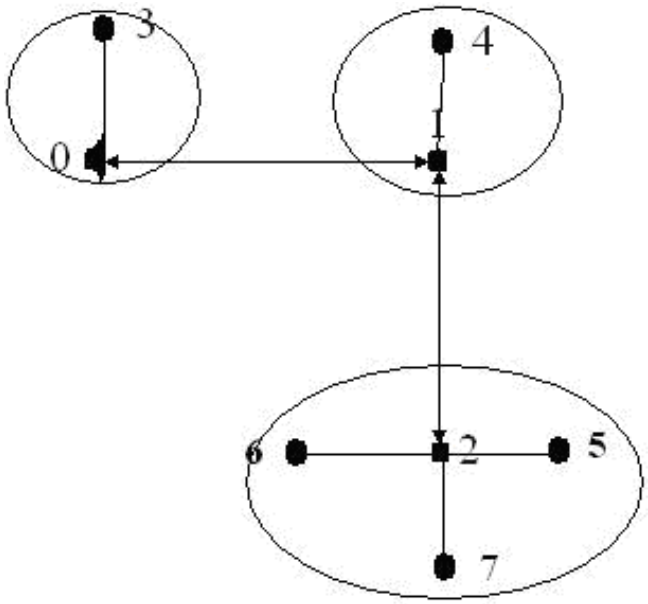


FIG. 5.4. *Network structure for case.*

The simulation was developed in the TOSSIM tool, and the TelosB platform was selected, including the CC2420 transceiver. Because the characteristics given of the case, with fixed nodes to implement the routing protocols, a routing fixed table algorithm was implemented, it is presented in table 5.1.

TABLE 5.1
Routing Table.

Source node	Destination node
2	1
1	0
0	

In the simulation was considered the most critical case, where all mobiles nodes are connected all time to the farthest sub-network from the main node.

Times obtained in sending 2 Bytes from all patients to the main node (node 0), are presented in table 5.2. The $data_1$ indicate that transmitting the message from a mobile node to the main node was over; the $data_2$ indicate that the corresponding node has not received the ACK. Times obtained demonstrated that it is possible to fulfill the constraint of 3s, for the transmission of patient status from a mobile node to the main node, which was imposed by the case analyzed.

6. Temporal Behavior of Networked Control Systems Over IEEE 802.15.4. Consequence to the increasing complexity of control systems most of activities have been distributed over different nodes, which control loops are closed through a communication network, these systems are called Networked Control Systems (NCS). The implementation of NCS also reduces the impact of failures in a system component and facilitates the diagnosis, maintenance and traceability processes.

Since the mobility of the elements that constitute the nodes in industrial processes is very low, the applications in this sector do not demand the strict use of wireless networks, for this reason in most cases wired networks are used, this is also consequence of reliability of wired networks and the ability to support transmission periods smaller than wireless networks. However, the development of new applications on wireless sensors and actuators networks (WSAN) will allow integrating wired and wireless networks to increase the applications flexibility and reliability, at the same time its impact on implementation reduction cost is significant.

This section presents the performance analysis of a NCS, using the MAC algorithm CSMA/CA implemented in IEEE 802.15.4 protocol. The generic diagram of the NCS considered in this work is presented in figure 6.1, which regulates the output signal in a second order system implemented with operational amplifiers. It has three types of nodes:

TABLE 5.2
Time in sending 2 Bytes from all patients to the main node.

Source node	Receiver node	Time (s)		Source node	Receiver node	Time (s)
6	2	78.309		0	$1(ack)_1$ - 5 ends	.613
4	2	.320		7	2 Rtx_2	.684
7	2	.333		3	2 Rtx_2	.684
2	6(ack)	.344		2	1 Rtx moving 4_2	.684
5	2 Rtx_2	.355		1	2 (ack)	.701
2	1 (moving frame from 6)	.380		7	2 Rtx_2	.714
3	2 Rtx_2	.380		1	0 (moving frame from 4)	.740
1	2(ack)	.397		2	7 (ack)	.746
1	0 (moving frame from 6)	.421		0	1 $(ack)_1$ - 4 ends	.764
5	2 Rtx_2	.421		2	1 (moving frame from 7)	.780
3	2 Rtx_2	.463		3	2 Rtx_2	.780
2	1 (moving frame from 5)	.486		1	2 (ack)	.792
7	2 Rtx_2	.486		3	2 Rtx_2	.807
2	5(ack)	.488		2	3 $(ack)_2$	.825
0	$1(ack)_1$ - 6 ends	.488		2	1 (moving frame from 3)	.860
1	2(ack)	.500		2	1 Rtx moving 3_2	.877
4	2 Rtx_2	.513		1	0 (moving frame from 7)	.877
3	2 Rtx_2	.524		0	1 $(ack)_1$ - 7 ends	.913
7	2 Rtx_2	.535		2	1 Rtx moving 3_2	.929
2	4(ack)	.547		1	2 (ack)	.949
2	1 (moving frame from 4)	.581		1	0 (moving frame from 3)	.979
1	0 (moving frame from 5)	.589		0	1 $(ack)_1$ - 3 ends	79.013

- Sensor, performs the measurement of the controlled signal and sends a frame with this information by the network.
- Controller-Actuator, receives a measure of the signal controlled, calculates the control action and acts over the manipulated signal in the system.
- Noise generator, generates network traffic.

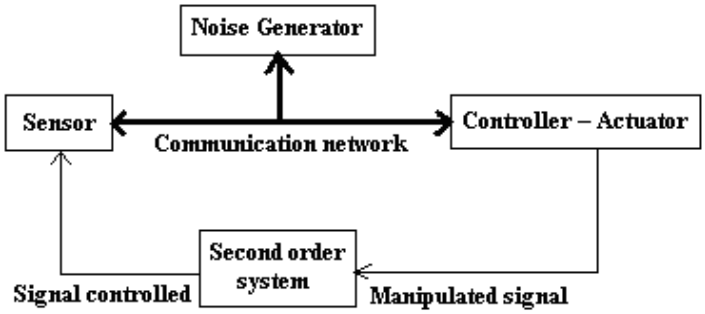


FIG. 6.1. *Generic diagram of the NCS considered.*

At the moment of design a high performance NCS is not always obtained similarity between experimental results and simulation, this is because imprecise models for analyzing and designing these systems are used, and for to make use of inadequate validation methods and platforms that not support the models used. There are several authors who have analyzed the performance and stability in NCS assuming network protocols with constant and variable delays, this shall also have made proposals to modified the control algorithms in order to respond to these effects, [18], [19], [20], [21]. An analysis of the performance of wired networks for control process is presented in [22]. In [23] the analysis of use 802.11b and Bluetooth networks in control systems is presented.

The transmission period in NCS is defined as the time between two consecutive transmissions, and is measured on- line for each communication segment. In NCS the transmission time between sensor and controller nodes can be periodic or aperiodic, being affected by delays depending on the Medium Access Control (MAC) protocol of the communication network, communication errors, Jitters and tasks scheduler.

The difficulty in the analysis and design of NCS is consequence of delays in the feedback control loops, because the behaviour of each component can affect the performance of control algorithms. Depending on the magnitude and variability of delays the performance of control systems can be degrade and can even present stability problems.

In the case study considered in this work, figure 6.1, the controller and actuator are in the same node (Controller-Actuator), so a single delay in the feedback control loop is considered, τ, which includes the processing time in the Sensor node, the network transmission time and the processing time in the Controller-Actuator node. The system was modelled by:

$$G_{p(s)} = \frac{20.3759}{s^2 + 3.497s + 21.73} \tag{6.1}$$

The regulator algorithm was designed as a PID algorithm using the Ziegler-Nichols closed-loop method, whose transfer function is:

$$G_{c(s)} = \frac{0.8909s^2 + 3.2322s + 20.201}{s} \tag{6.2}$$

It's representation in discrete time is: $u_k = u_{k-1} + q_0 e_k + q_1 e_{k-1} + q_2 e_{k-2}$; $q_0 = kp + \frac{kd}{T_m}$; $q1 = -kp - \frac{2kd}{T_m} + kiT_m$; $q2 = \frac{kd}{T_m}$

As a first approximation to analyzing the case study τ was considered constant. Using a Pade second order polynomial to model the delay, an approximation to the stability region for the feedback control system was found [24], [25], figure 6.2. It gives information of sampling period (T_m) and τ values for the system stability. This information can be used to choose the periods and deadlines to implement the control system.

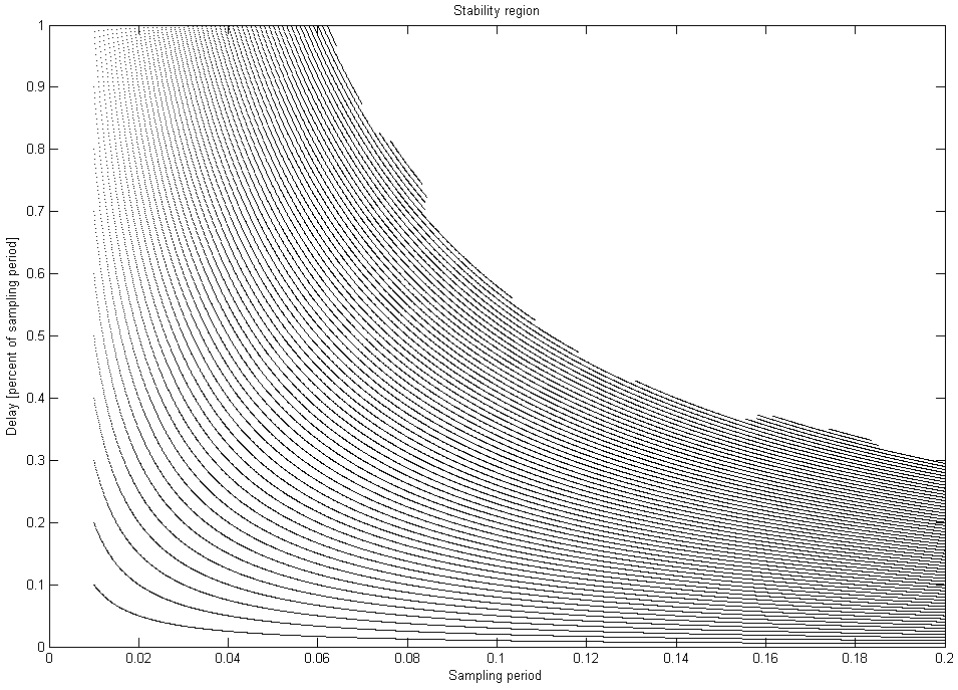


FIG. 6.2. *Stability region.*

Figures 6.3, 6.4 presents the control system output for different values of T_m and τ. and . It is possible to see how increasing the sampling period the system is more sensitive to delays, and although the system is stable to the considered values there is a large degradation in the performance of the control system, which could not ensure the desired performance in some cases.

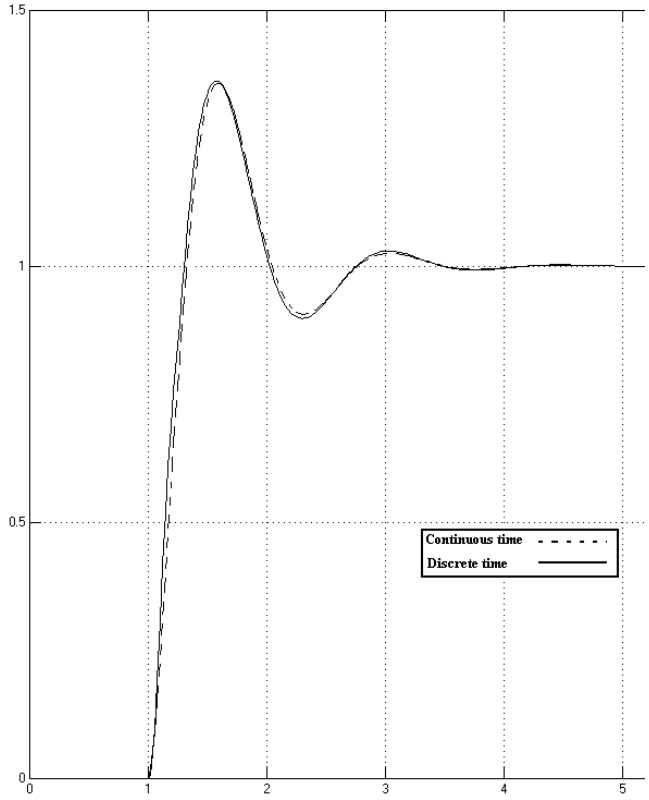


FIG. 6.3. *Control system output in continuous and discrete time for* $T_m = 23ms$, $\tau = T_m$.

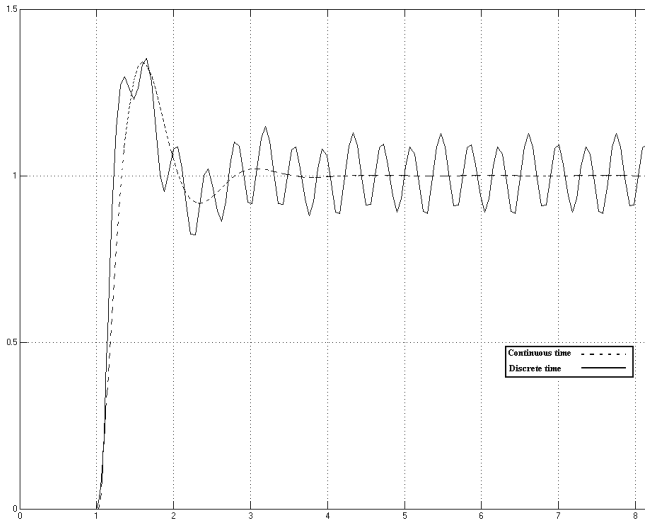


FIG. 6.4. *Control system output in continuous and discrete time for* $T_m = 57ms$, $\tau = 0.5T_m$.

To analyze the NCS considered the Truetime simulator was used [26]. Only the effect of MAC protocol was considered and the time processing in the nodes was ignored. The simulation parameters were:

- Sampling periods for the controlled signal: 50ms.
- Synchronization by events between nodes Sensor and Controller-Actuator.
- Data rate: 250 kbps.
- Frame size of 82 bits. Enough to send the measure of a variable.
- Two Noise generator nodes were implemented, with periods of 700 μ s and 900 μ s, and frame size of 82 bits each one.

The simulation results are presented in figure 6.5. It is possible to see how the time for sending information by CSMA/CA is variable and unbounded. Delays in the feedback control loop do not affect significantly the performance of the system, figure 6.6.

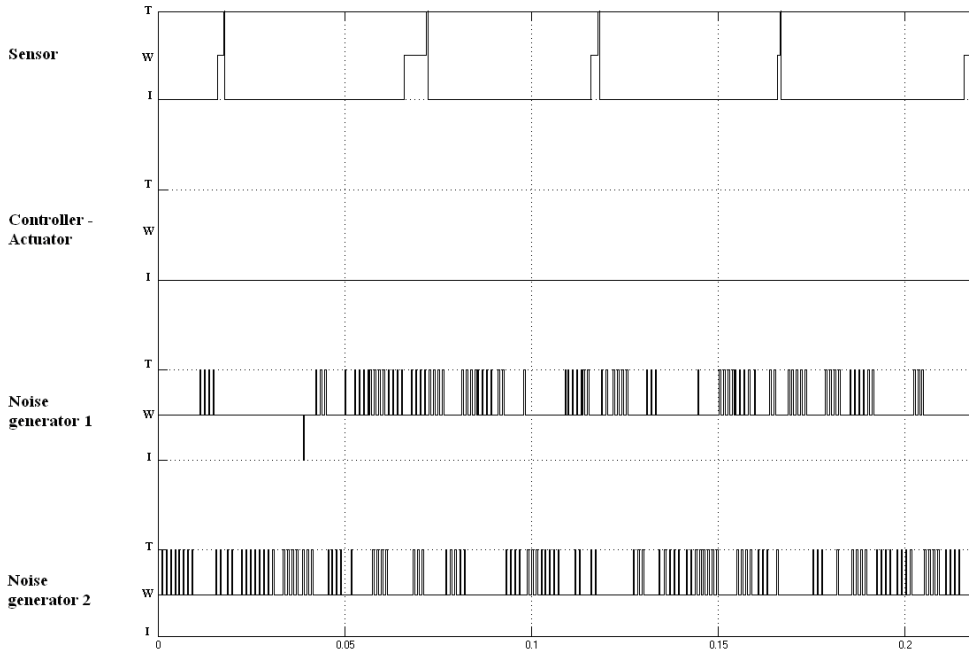


Fig. 6.5. *Network schedule. Levels T, W, and I represents Transmitting, Waiting and Idle states in every node.*

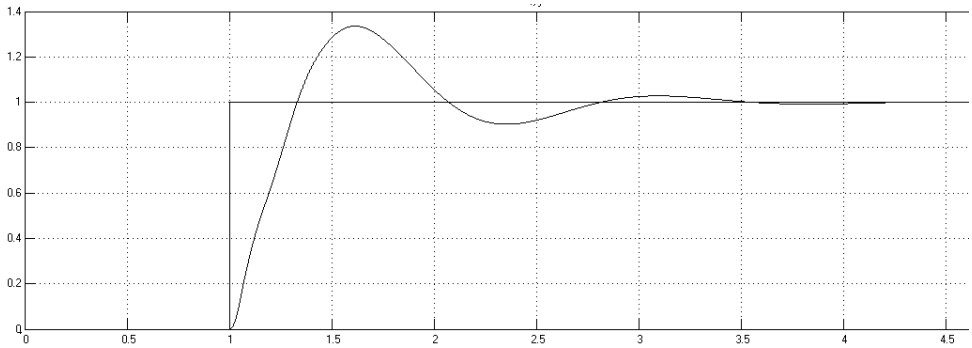


Fig. 6.6. *System response.*

The network was saturated by using transmission periods smaller than previous cases in the Noise generator nodes, figure 6.7, which was noted than despite loss of information transmitted by the Sensor node as a consequence of collisions with noise frames, the Sensor can retransmit several times before next transmission period, then the Controller-Actuator node get the measure before a critical τ (according to the stability region) and the system is stable, but the performance of control systems is degraded, figure 6.8.

7. Implementation of NCS. The implementation of the NCS was developed on IEEE 802.15.4 mode CSMA/CA. To develop the Sensor and Controller-Actuator nodes boards with the CC2430 processor were used, also the abstraction levels HAL and OSAL from Texas Instruments were used to access the hardware and to implement tasks. Two Noise generator nodes were implemented by MACdongle devices. The configuration was as follows:

- The Sensor node sends a frame every 60 ms. This period was selected because it was experimentally observed that lower values for the period was not stable as a consequence of computing in the node.
- In the Controller-Actuator node an event is generated for every message received from the Sensor node. During its attention the control output is calculated and acts on the system. Experimentally an interval

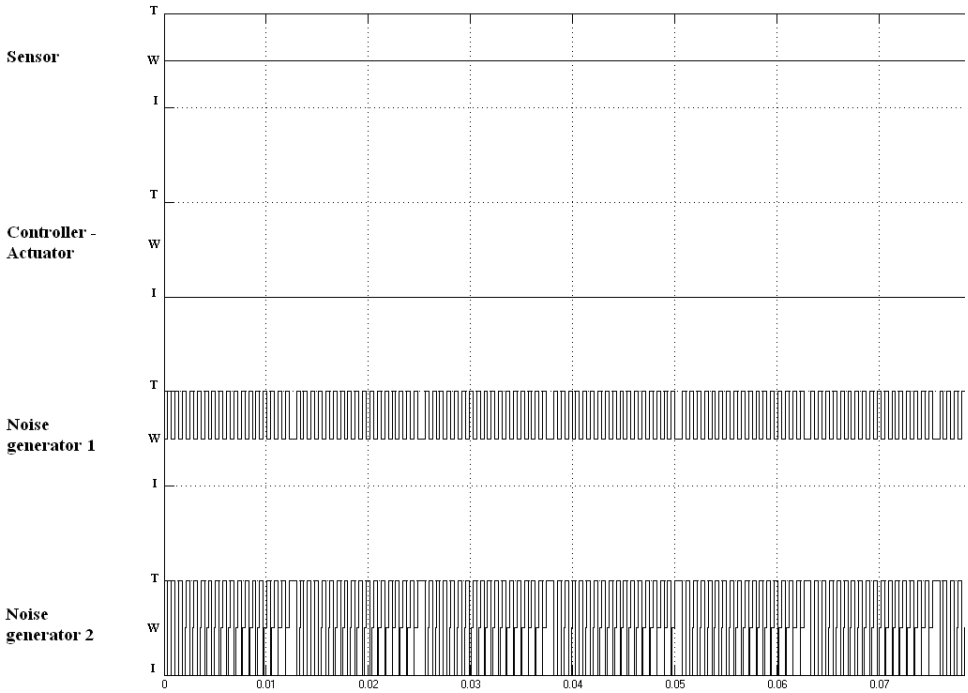


FIG. 6.7. *Network schedule with network saturated. Levels T, W, and I represents Transmitting, Waiting and Idle states in every node.*

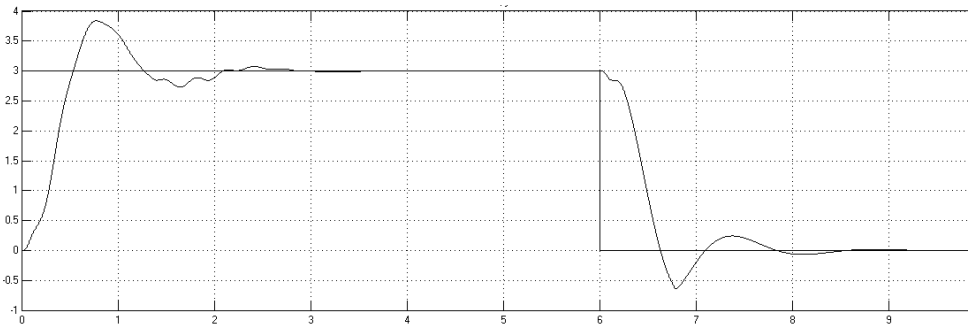


FIG. 6.8. *System response for network saturated.*

time between 12 ms and 16 ms was obtained from the time start measurement in the Sensor node until Controller-Actuator node acts on the system.

- The Noise generators nodes send a frame every 30ms.
- The size of the frame is 256 bits and the data rate is 250 kbs. Then the time of sending a frame is 1.024 ms.
- The nodes were distributed in an area of 1 m^2.

The Truetime simulations and experimental results are shown in figures 7.1, 7.2, 7.3 and 7.4.

As a result to the delay generated to frames from Sensor node, as a consequence of collisions in the network with frames sending from Noise generators nodes, in figure 7b can be seen than the transmission period is variable and it is not bounded. Moreover, the feedback delay is small compared to the dynamics of the system, and therefore the system is not significantly disturbed.

Experimental results are close to those gotten by simulation in Truetime.

8. Conclusions. From the study it can be concluded that developments on specific technologies and applications in this area are still emerging, and developing them will enable the growing of great social impact new applications.

About the system for detecting heart arrhythmias:

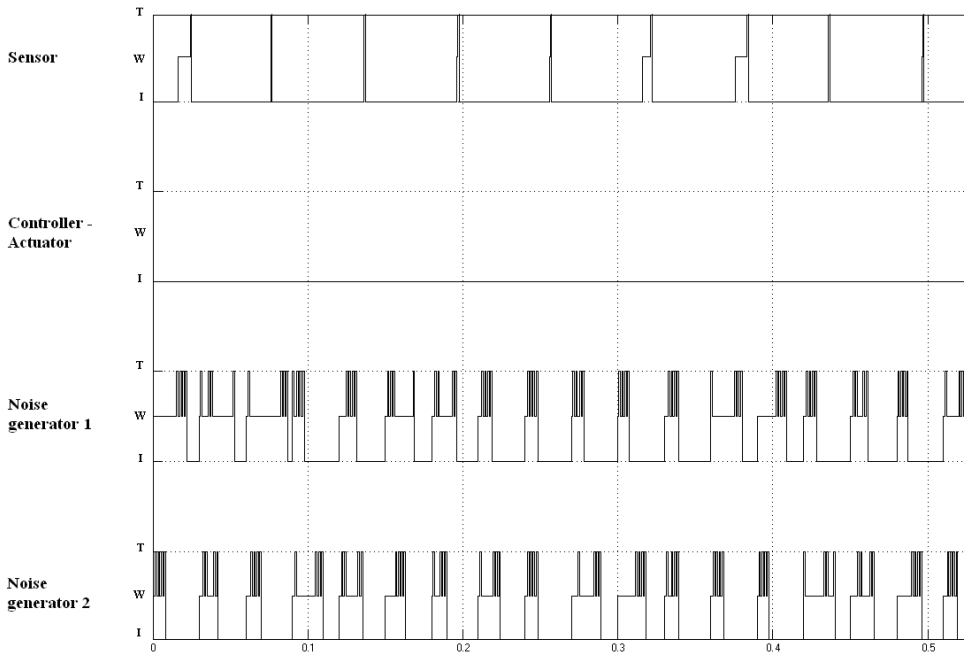


FIG. 7.1. *Simulation results of Network schedule.*

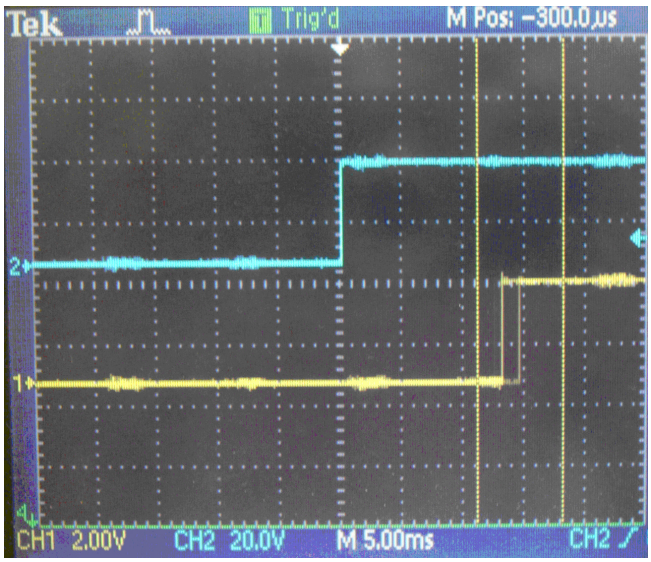


FIG. 7.2. *Experimental results of message time delay.*

- The proposed architecture considers the constraints of application field, allowing to find optimal solutions to the challenges in network and nodes designing, and will facilitate the development and validation of applications. It makes possible too the cooperation between levels of the network architecture to choose between different operations modes depending on QoS indexes.
- It is noted as the routing algorithm based on fixed tables supported by IEEE 802.15.4, fulfils the time requirements of these applications. Also as MSP430 architecture presents a good performance for the implementation of the case considered.

The impact of delays in the feedback loop of NCS was analyzed with formal methods, simulation and experimental results, which concludes:

- The technology considered in this paper can be used in control applications, where transmission times are not very demanding, particularly we propose use in cases with transmission periods upper then 100 ms.

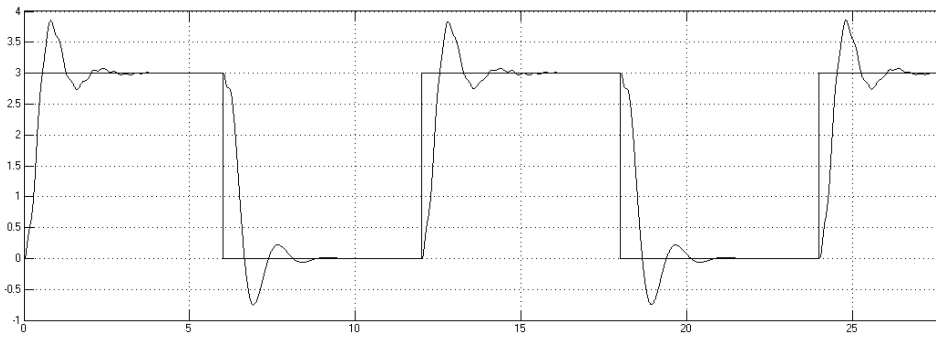


Fig. 7.3. *Control system output implemented on 802.15.4 mode CSMA/CA with two noise generators operating every 30 ms.*

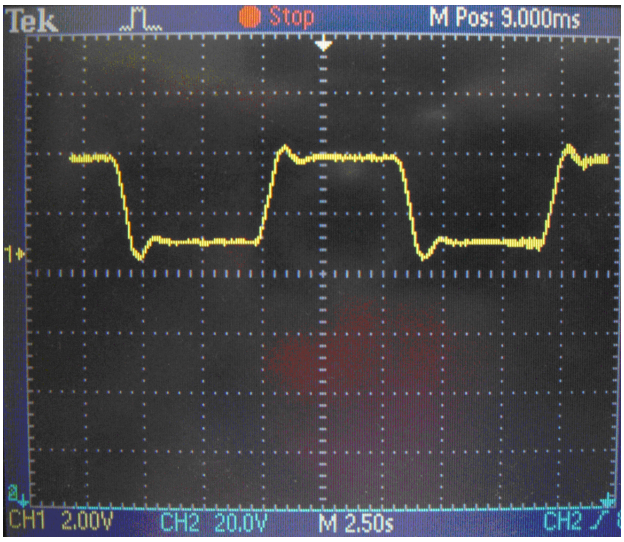


Fig. 7.4. *Experimental values of control system output.*

- For big sampling periods the system is more sensitive to delays in the feedback loop, so it is important to maintain delays bounded.
- To control systems with transmission period less than 100 ms is recommended the use of wired networks, which is possible to get transmission periods smaller and stable than IEEE 802.15.4 mode CSMA / CA.
- There is a great similarity between the results using formal methods, simulation and physical implementation of the system. Then is possible to conclude than the analysis methods (stability region) used and simulation environments as Truetime, allow a fast analysis and reliable of such applications, which facilitates the design process of real NCS.

The future work proposed is the performance analysis of different task schedulers and routing protocols on the presented architecture, and a cooperation strategy between them to minimize power consumption.

REFERENCES

[1] S. Shivakumar, S. S. Iyengar, *Real-Time Sensor-Actuator Networks*, International Journal of Distributed Sensor Networks, Pages: 17–34, 2005.
[2] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, Y. Fun Hu, *Wireless Sensor Networks: a survey on the State of the Art and the 802.15.4 and ZigBee Standards*, Computer Communications, Vol. 30, No. 7, Pages: 1655–1695, 2007.
[3] S. Ananthram, Z. Qing, H. Yao-Win, T. Lang, *Wireless Sensor Networks*, WILEY, Pages: 251–343, 2007.
[4] P. J. Marron, D. Minder, and the Embedded WiSeNts Consortium, *Embedded WiSeNts Research Roadmap*, 2006.
[5] J. L. Hill, *System architecture for wireless sensor networks*, Doctoral thesis, University of California, Berkeley, 2003
[6] D. Culler, D. Estrind, M. Srivastava, *Overview of Sensor Networks*, IEEE Computer, Pages: 41–49, August, 2004.
[7] B. Son, Y. Her, J. Kim, *A Design and Implementation of Forest-Fires Surveillance System based on Wireless Sensor Networks for South Korea Mountains*, International Journal of Computer Science and Network Security. Vol. 6 No. 9B, September, 2006.

[8] L. Yu, N. Wang, X. Meng, *Real-time Forest Fire Detection with Wireless Sensor Networks*, Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing, 23–26 September, 2005.
[9] P. K. Haneveld, *Evading Murphy: A sensor network deployment in precision agriculture*, Technical Report, June 28, 2007.
[10] N. Wang, N. Zhang, M. Wang, *Wireless sensors in agriculture and food industry—Recent development and future perspective*, Computers and Electronics in Agriculture. Vol. 50, N. 1, Pages: 1–14, January 2006.
[11] A. Baggio, *Wireless sensor networks in precision agriculture*, Workshop on Real-World Wireless Sensor Networks. REALWSN'05. Sweden, June 20-21, 2005.
[12] Benny P., L. Lo, G. Z. Yang, *Key technical challenges and current implementations of body sensor networks*, Department of Computing, Imperial College London, UK. http://www.doc.ic.ac.uk/ benlo/ubimon/BSN.pdf.
[13] J. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, A. Wood, *Wireless Sensor Networks for In-Home Healthcare: Potential and Challenges*, High Confidence Medical Device Software and Systems (HCMDSS) Workshop, Philadelphia, PA, June 2005.
[14] T. Norgall, *Body Area Network BAN—a Key Infrastructure Element for Patient-Centric Health Services*, ISO TC215/WG7/IEEE 1073 Meeting, Berlin, May 2005
[15] J. Pan, W. J. Tompkins, *A Real-Time QRS Detection Algorithm*, IEEE Trans. Biomed. Eng., vol. BME-32, Pages: 230–236, 1985
[16] W. J. Tompkins, J. G. Webster, *Desing of microcomputer-based medical instrumentation*, New Jersey: Prentice-Hall, Pages: 396-397, 1981.
[17] P. Pillai, K. G. Shin, *Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems*, Proceedings of the eighteenth ACM symposium on Operating systems principles. Banff, Alberta, Pages: 89–102, Canada, 2001.
[18] G. C. Walsh, H. Ye, L. Bushnell, *Stability analysis of networked control systems*, submitted to American Control Conference, June 1999.
[19] W. Gregory, B. Octavian, B. Linda, *Asymptotic Behavior of Networked Control Systems*, Submitted to Control Applications Conference, International Conference on Control Applications. Hawaii, USA, August 22–21, 1999.
[20] T. C. Yang, *Networked control system: a brief survey*, IEE Proc. Control Theory Appl., Vol. 153, No. 4, July 2006.
[21] J. P. Hespanha, Y. Xu, *A Survey of Recent Results in Networked Control Systems*, Proceedings of the IEEE, Vol. 95, No. 1, January 2007.
[22] F. Lian, J. R. Moyne, D. M. Tilbury, *Performance Evaluation of Control Networks: Ethernet, ControlNet, and DeviceNet*, IEEE Control Systems Magazine February, 2001.
[23] D. Hristu-Varsakelis, W. S. Levine, *Handbook of Networked and Embedded Control Systems*, Pag: 677–720, Birkhäuser 2005.
[24] F. Gene, P. David, W. Michael, *Digital Control of Dynamic Systems*, Pag: 39–41. Ed Addison-Wesly, 1990
[25] W. Zhang, *Stability Analysis of Networked Control Systems*, PhD Thesis, Department of Electrical Engineering and Computer Science, Case Western Reserve University. Pag: 99–3103, August 2001.
[26] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K. Arzen, *How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime*, IEEE Control Systems Magazine, 2003.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 307–324. http://www.scpe.org
ISSN 1895-1767

COMBINATION OF LOCALIZATION TECHNIQUES FOR MOBILE SENSOR NETWORK FOR PRECISE LOCALIZATION*

HA YOON SONG†

Abstract. A precise localization is required in order to maximize the usage of Mobile Sensor Network. As well, mobile robots also need a precise localization mechanism for the same reason. In this paper, we showed a combination of various localization mechanisms for precise localization in three different levels. Localization can be classified in three big categories: wide area and long distance localization with low accuracy, medium area and distance localization with medium accuracy, and small area short distance localization with high accuracy. In order to present localization methods, traditional map building technologies such as grid maps or topological maps can be used. We implemented mobile sensor vehicles and composed mobile sensor network with three levels of localization techniques. Each mobile sensor vehicles act as a mobile sensor node with the facilities such as autonomous driving, obstacle detection and avoidance, map building, communication via wireless network, image processing, extensibility of multiple heterogeneous sensors, and so on. For localization, each mobile sensor vehicle has abilities of the location awareness by mobility trajectory based localization, RSSI based localization and computer vision based localization. With this set of mobile sensor network, we have the possibility to demonstrate various localization mechanisms and their effectiveness. In this paper, the result of computer vision based localization, sensor mobility trail based localization and RSSI based localization will be presented.

Key words: localization, mobile sensor network, RSSI, dead-reckoning, computer vision

1. Introduction. The researches on Mobile Sensor Network (MSN) have been plenty worldwide. For MSN, there could be a lot of valuable application with attached sensors as well as capabilities such as locomotion, environmental information sensing, dead-reckoning, and so on. For such applications, usual requirements have been acknowledged with localization of each sensor node and formation of the whole sensor network. In this research we are going to discuss about localization techniques for Mobile Sensor Vehicle (MSV) which can compose MSN. In addition, we will discuss a construction of MSN as well as required functionalities of each MSN. For the precise localization we may analyze human actions for localization. For long distance and huge area localization, humans call to their counterpart and identify counterpart's location by talking each other. From the conversation, only a rough location can be identified. Thus we guess long distance localization only allows rough, inaccurate information of location but it is sufficient to confine a region for more precise localization. For medium distance and medium area localization, human moves by transportation methods but eventually walks in order to localize. While walking, humans build a conceptual map for the local geographic information or they already have knowledge around the area, i. e. they already built maps. This sort of medium distance localization requires relatively more precise localization information than long distance localization as precise as, at least, for walking, i. e. autonomous driving.

For short distance and small area localization, humans detect counterparts by use of visual or aural information, i. e. they find their friends by their eyes. This sort of localization deduces precise information than other two sort of localization.

Thus, we can conclude and mimic the human localization with mobile sensor networks. Even though it depends on techniques and environments of the usage of MSN, we can categorize the localization technique according to its area or distance.

There have been variety forms of Mobile Sensor Nodes which utilizes various techniques of localizations such as RSSI, GPS, Raider, Laser, Camera, and so on [1] [2]. One of the most prominent one, an RSSI based localization, usually measures radio signal strength and it works well with popular network devices. Moreover, an 802.11 device based software approach can be realize easily as we did in this paper. However, RSSI method is prone to be fragile with a presence of obstacles or so which will diminish or attenuate radio signal strength. In a short distance, RSSI signals usually is too high that nullify accurate localization thus it is good for long distance, low accurate localization.

By mimicking human actions for localization we can choose RSSI for mobile sensors while wireless telephones for human and trajectory based tracking, so called INS (Inertial Nautical System), as human walking. Of course, map building techniques are required for MSV as well as humans. For human visual localization, we can choose

*This work was supported by the 2008 Hongik University Research Fund.

†Department of Computer Engineering, Hongik University, Seoul, Korea (hayoon@wow.hongik.ac.kr). This paper had been completed during author's sabbatical year of 2009 at Institute of Computer Technology, Vienna University of Technology, A-1040 Vienna, Austria (song@ict.tuwien.ac.at).

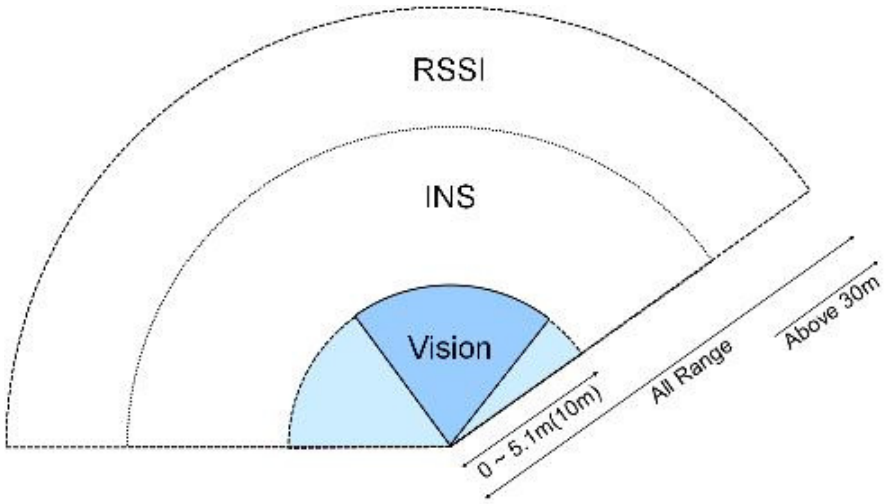


FIG. 1.1. *Categorization of Localization Techniques*

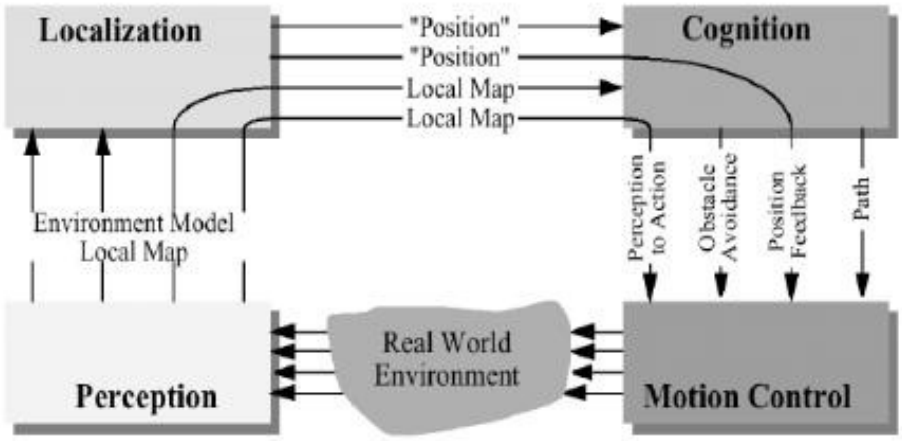


FIG. 1.2. *Relative Techniques for Localization*

a localization technique based on computer vision. Thus we conclude and categorize localization techniques in three big categories as shown in figure 1.1 so that we can choose proper localization mechanisms for its usage.

And in order to fulfill the localization, not only localization techniques but also techniques of cognition, motion control, and perception are tightly related as shown in figure 1.2. We must express idea of this paper in terms of these concepts of cognition, motion control, perception and localization.

This paper is organized as follows. In section 2 we will discuss localization method that have been researched. The following section 3 we will analyze the requirement for MSV, the hardware design of MSV, and equipments for localization, and we will discuss software capabilities of MSV software and will show software components to fully control our MSV including software for MSN itself, monitoring program, map building features, and other related topics. Then section 4 will shows the approaches of computer vision based localization for small area, short distance precise localization. In section 5, our approach and methodology for mobility trajectory based localization for medium distance localization will be discussed. In Section 6 RSSI (Radio Signal Strength Identification) based localization will be presented based on 802.11 devices with software modification. Finally section 7 will conclude this paper with possible future research topics.

2. Related Works. There have been a lot of researches regarding mobile sensor localization. In this section, we will discuss past researches and our idea concentrating techniques with RSSI, vision and trajectory-tracking. This works are not restricted on mobile sensors only but also related to robot technology.

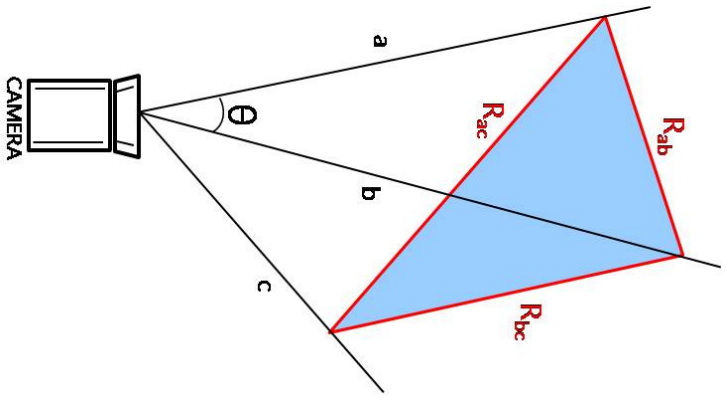


FIG. 2.1. *Calculation of distance from Camera Lens to Vertex of triangle ABC*

2.1. RSSI based Localization. Radio Signal Strength Identification is one of the known solutions for distance measure. It requires wireless network device for mobile sensors and extra features.

We use 802. 11 network devices which have wide popularity. In addition we need communication between mobile sensors thus 802.11 networking devices are popular solutions for us. RSSI features for 802.11 device networks are required features in order to implement physical layer of CSMA/CA networking [14]. However RSSI based distance measure is very prone to radio signal attenuation and thus has low accuracy. And it has some restriction that once it is a data transfer modes, it cannot switched to API mode instantly. It implies the restricted realtimeness for RSSI based localization. Manufactures of 802.11 devices usually provide their arbitrary method for RSSI [11]. In this paper, we will demonstrate our MSV successfully does long distance, low accurate localization only with commercial 802.11 devices and networking software embedded on MSVs.

2.2. Computer Vision based Approach. There are very few researches on localizations by use of computer vision technology. There have been the previous results regarding mobile sensor vehicle control, obstacle detection and so on.

Matsummoto et al. [15] used multiple cameras in order to control mobile robots. In their research, cameras are installed on their working space instead of mobile vehicle itself. Their whole system is consisted of mobile robots and multiple cameras and this helps the search of proper path of robots. Keyes et al. [16] researched various camera options such as lens type, camera type, camera locations and so on. They also used multiple cameras to obtain more precise information.

In this paper we will provide MSV with multiple cameras in order to accomplish short distance, high accurate localization. However, a single MSV cannot locate its location precisely. The ultimate localization can only be done with the cooperation of nodes in MSN.

The first requirement for localization is to identify the location of colleague MSV as a base point. For this purpose, we prepared three facilities for each MSV. Each MSV can estimate its location by trajectory trail. Moreover, each MSV can identify other colleague MSV with their infrared LED signal. In addition, this location information can be communicated by wireless network device equipped with each MSV.

Of course, a camera or a set of cameras are installed on an MSV in order to identify colleague MSVs. This set of cameras has infrared filters in order to diminish the effect of extra light noise in operating environment.

2.2.1. Location Determination Problem. With a set of camera, the required information for localization is collected from the view of cameras. For example, an infrared LED light can be a parameter to calculate the colleague's location. In this research, we applied two previous results. The first one is Sample Consensus(RANSAC) Method [5] and the second one is PnP Method [6] [7].

For RANSAC method, because of least square method, there is no possibility of wrong computation with gross error value. This is the major reason why we applied RANSAC method. In order to solve the problem of converting 3-dimensional view to 2-dimensional camera image, which has lost distance problem, we applied perspective-3-point (P3P) problem.

Figure 2.1 shows the basic principle of P3P problem. The gray triangle is composed by infrared LED installed on each MSV. Points A, B, C stand for each infrared LEDs and these vertices compose a triangle. The distance R_{ab}, R_{bc}, R_{ac} is known constants. From figure 2.1 we can drive the following very well-known

FIG. 3.1. *Driving Mechanism Outlook*

mathematical equation as shown in equation 2.1. The equation 2.1 is in closed form. The number of solutions from these equations will be up to eight. However, there are up to four positive roots.

$$\begin{aligned} {R_{ab}}^2 &= a^2 + b^2 - 2ab \cdot \cos\theta_{ab} \\ {R_{ac}}^2 &= a^2 + c^2 - 2ac \cdot \cos\theta_{ac} \\ {R_{bc}}^2 &= b^2 + c^2 - 2bc \cdot \cos\theta_{bc} \end{aligned} \tag{2.1}$$

With this P3P based method, we can only measure distances between observer and observed. For precise localization, we must identify angle of MSVs as well. Our MSV is equipped with digital compass in order to identify the angle of MSV based on magnetic poles. As predicted, digital compass also has its own error in angle measurement but is tolerable.

2.3. Autonomous Driving Robot and Dead-Reckoning. This sort of localization is usually due to military area. For example DARPA, USA invests on unmanned vehicle, and their aim is about 30% of army vehicle without human on board controller. Stanley by Stanford university [17], which earned first prize in competitions, are equipped with GPS, 6 DOF gyroscope and can calculate the speed of driving wheels. Those sensors information can be combined to locate the position of their unmanned vehicle. They used computer vision system with stereo camera and single camera, and laser distance meter, radar in order to get environmental information. Sandstorm from CMU [18] is equipped with laser distance meter as a major sensor. Topographical model can be obtained by laser lines and the speed of car can be calculated by the density of laser line. Gimbal on their vehicle can install long distance laser scanner with seven laser sensors. Shoulder-mounted sensors can calculate height information of topography. Two scanners on bumpers can obtain obstacle information. Long distance obstacles can be identified by radar.

Our MSV are equipped with RSSI devices, stereo cameras and other sensors for dead-reckoning. Apart from the examples of locomotive robots, these equipments are for accurate localization.

3. Mobile Sensor Vehicle. We developed MSV in order to experiment our localization method in real environment. Various versions of MSV are designed and implemented. The localization functions implemented on MSV are as follows:

- Long distance low accuracy localization by RSSI
- Medium distance medium accuracy localization by dead-reckoning tracking
- Short distance high accuracy localization by Stereo camera with computer vision.

In the following subsection we will discuss hardware and software of MSV respectively.

3.1. Hardware. MSV is actually a mobile sensor node for MSN. Each MSV can move autonomously and can identify obstacles. They can communicate each other by 802.11 networking devices. The chassis of MSV are composed of aluminum composite with high durability and lightweight. The main driving mechanism

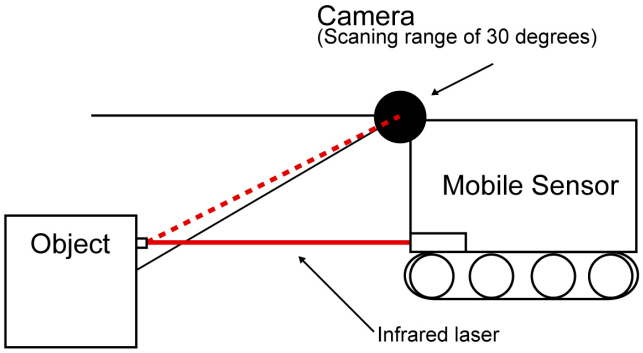


Fig. 3.2. *Obstacle Detection Mechanism*

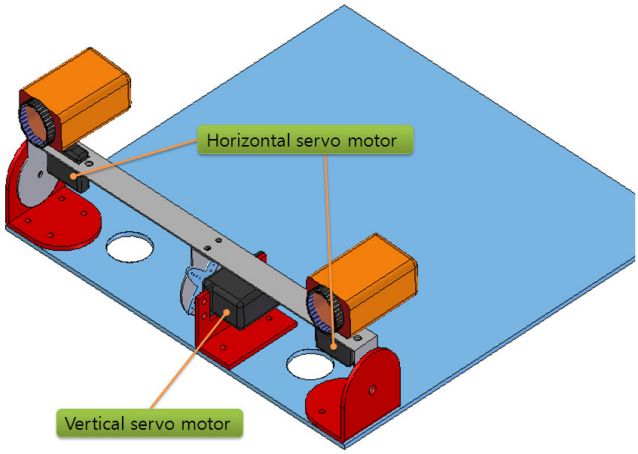


Fig. 3.3. *Stereo Eye System*

is caterpillar composed of three wheels, L-type rubber belt, gears as shown in figure 3.1. The adoption of caterpillar is for minimization of driving errors. There are a lot of rooms to install additional sensor hardware. With digital compass equipped on the top of MSV, the accurate vehicle location can be sensed. This angular information can help exact localization of MSVs. The design concepts of MSV are as follows.

- Autonomous mobility
- Extensibility of equipped sensors
- Precise movement and mobility trail

And MSV characteristics as a node of mobile sensor network are as follows:

- Self identification and colleague identification with various methods
- Wireless communication
- Digital Compass

For autonomous driving, MSV must identify obstacles and avoid them. We use an infrared laser and cameras with infrared filter. IR laser is constantly lighting in parallel to round. Camera looks down grounds in a degree of 30 which is determined by experiments. The concept of this obstacle detection is depicted in figure 3.2. Obstacle reflects IR laser and sensed by camera [5, 7]. The obstacles with reflected IR will be detected as white lines. This obstacle detection will be used by local map building as shown in section 5.1.

For short distance obstacles within the dead angle of camera, ultrasonic sensors are located under the MSV and in front of MSV. For computer vision based localization, MSVs are equipped with stereo eyes as shown in figure 3.3. Three servo motors can control two cameras independently. This stereo camera system can be used not only for localization but also for obstacle detection with diminished dead angle. There are three infrared LEDs mounted in the front of MSV. These LEDs are for computer vision based localization.

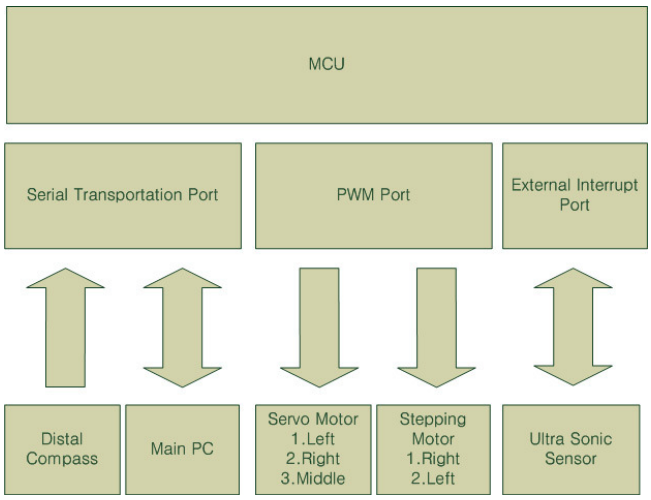


FIG. 3.4. *Hardware Structure*

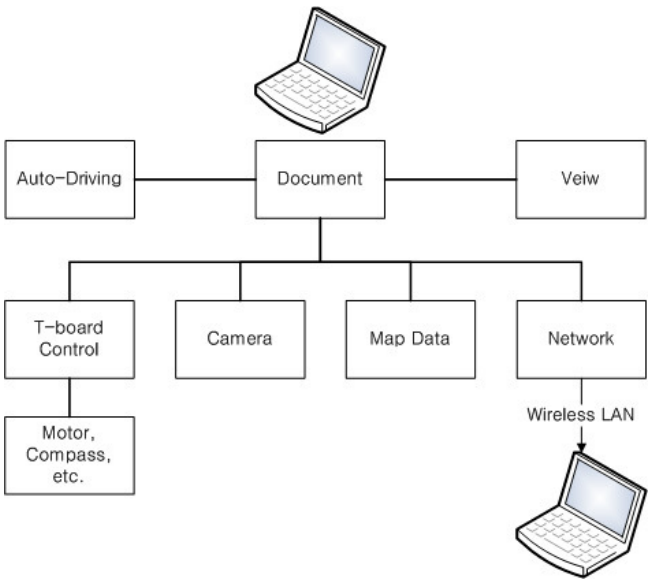


FIG. 3.5. *Software Structure*

For hardware construction, we need micro controller unit, a serial communication port, a PWM port, an interrupt port in order to control motors and communicate with sensors. Figure 3.4 shows the conceptual structure of MSV hardware.

3.2. Software. Software for MSV operations is required in a form of embedded software. Figure 3.5 shows required facility and their structure for MSV software.

Total part of software can be divided into five categories. One of the roles of software is to convert sensor information into driving information. Information for driving can be obtained via serial communication from T-board (MCU) with driving information and angular information.

The location of MSV is constantly updated with the moving distance and updated angle. Camera class provides obstacle information as well as basic information for map building. Map building class builds a map with the information from T-board class and camera class. These maps are required for autonomous locomotion and localization. Network class provides networking functionalities between MSVs.

We implement core software based on multi threads. There is document class, which provides organic data flow between classes. Thus the major role of MSV software is as follows.

- Autonomous driving
- Motor control and driving distance identification

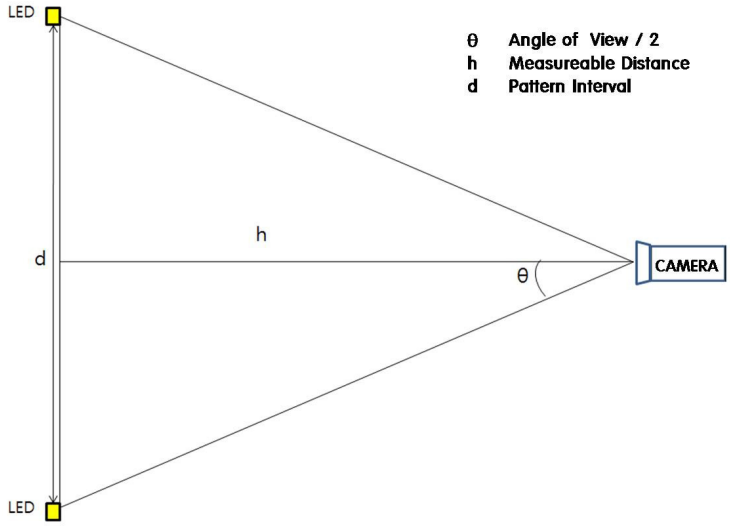


FIG. 4.1. *Correlation of LED pattern interval and measurable minimum distance*

FIG. 4.2. *Frond view of MSV*

- Communication with MCU (T-board)
- Obstacle detection
- Map building
- Internetworking
- User interface dialog (Monitoring Program)

Monitoring program is a user interface between MSV and user. Monitoring program shows MSV condition, camera view, driving information, map built, and other sensor information. It also provides manual operation functionality of MSV.

4. Computer Vision based Accurate Localization.

4.1. Sensor Equipment and Experimental Environment. Each MSV has a set of Infrared LED (IR-LED) in a form of triangle and the lengths of edges are all 30 centimeters. The IR lights from these LEDs can be viewed by stereo camera system from other colleague MSV. The stereo eye system as shown in figure 3.3 has two cameras. Three servo motors controls two stereo eyes vertically and horizontally.

TABLE 4.1
IR-LED specifications

MODEL NO.	Half Angle	Peak Wavelegnth
SI5315-H	±30°	950nm
OPE5685	±22°	850nm
OPE5194WK	±10°	940nm
TLN201	±7°	880nm
EL-1KL5	±5°	940nm

The stereo cameras are equipped with IR filters. The front view of MSV for these equipments is as shown in figure 4.2. Three IR-LEDs forms a triangle and a stereo camera system are also presented.

With fixed length of triangle edges, i. e. interval between IR-LED, is fixed by 30 centimeters. Therefore by using P3P method, the distance between camera and MSVs with IR-LED triangle can be calculated. Embedded software for each MSV has a realtime part for P3P solution. The software also shows the image from stereo camera as a part of P3P solution.

The ideal situation starts by estimating the angle between two cameras. Once camera direction is fixed, we can estimate angles between tracked object and cameras, however, MSV can move every direction which causes difficulties to measure that angle. Moreover, if these cameras have pan-tilt functionalities, it is impossible to measure such an angle in real time.

Another method with P3P technique is to assume the distance to the object. The distance to object and the scale of triangle in camera view is proportional inversely thus the size of LED triangle can be a starting point to estimate the distance to obstacles. We decided to standardize the reduced scale of LED triangle in order to estimate distance to objects. The basic concept of this method is depicted in figure 4.1 and will be discussed further in the subsection 4.3.

This approach has limits of camera visibility, i. e. objects beyond visibility cannot be identified. However, two other localization methods will be presented in the following sections for beyond sight localization. In addition with the help of digital compass, we can measure the direction of each MSV. The combination of this information can achieve short distance accuracy for localization.

4.2. Preliminary Experiment for Equipment Setup. We conduct preliminary experiment in order to choose optimal device for computer vision based localization. The first purpose of this experiment is to select the best LED in order to increase the range of localization. Our past result showed 250 centimeter of localization range however our aim is to enlarge the range to 400 centimeters or farther.

We choose five infrared light emitting diodes with typical characteristics. We first concentrated on the visible angle of LED lights since we assumed wider visible angle guarantees the clearer identification of LED light and more precise localization.

Table 4.1 shows the specifications of various IR-LEDs with visible angle and peak wavelength. The major reason why we choose those IR-LEDs are as follows:

- Smaller half angle of LED enables long distance tracking however increases invisibility from the side.
- Larger half angle of LED enables tracking from the side however decreases tracking distance.

With infrared filter equipped cameras we planned experiments to evaluate the LEDs for vision based localization. Table 4.2 show the result of visible distance and visibility of IR-LEDs. Twelve experiments have been made and average values are shown. From the specifications of IR-LEDs, 5 volts DC voltage is supplied for the experiment.

Among five IR-LEDs, two showed stable visibility and acceptable visibility distance. Between these two candidates, we finally choose the best LED of MODEL NO.SI5313-H since it has the widest half angle as well with reasonable visibility distance.

4.3. Main Experiments for Computer Vision based Localization. Figure 4.1 shows the relationship between LED triangle size (d) and distance from camera to LED triangle (h). The relation between d and h

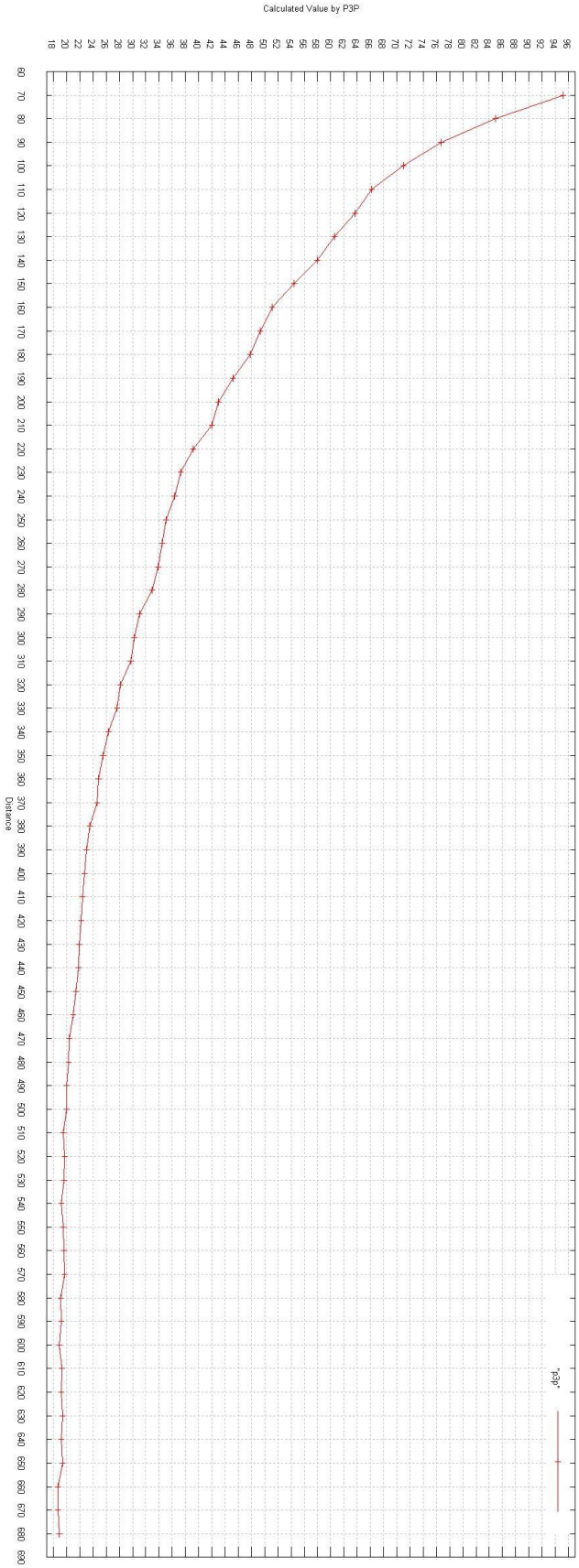


FIG. 4.3. *Relative size of triangle calculated by P3P on actual distance*

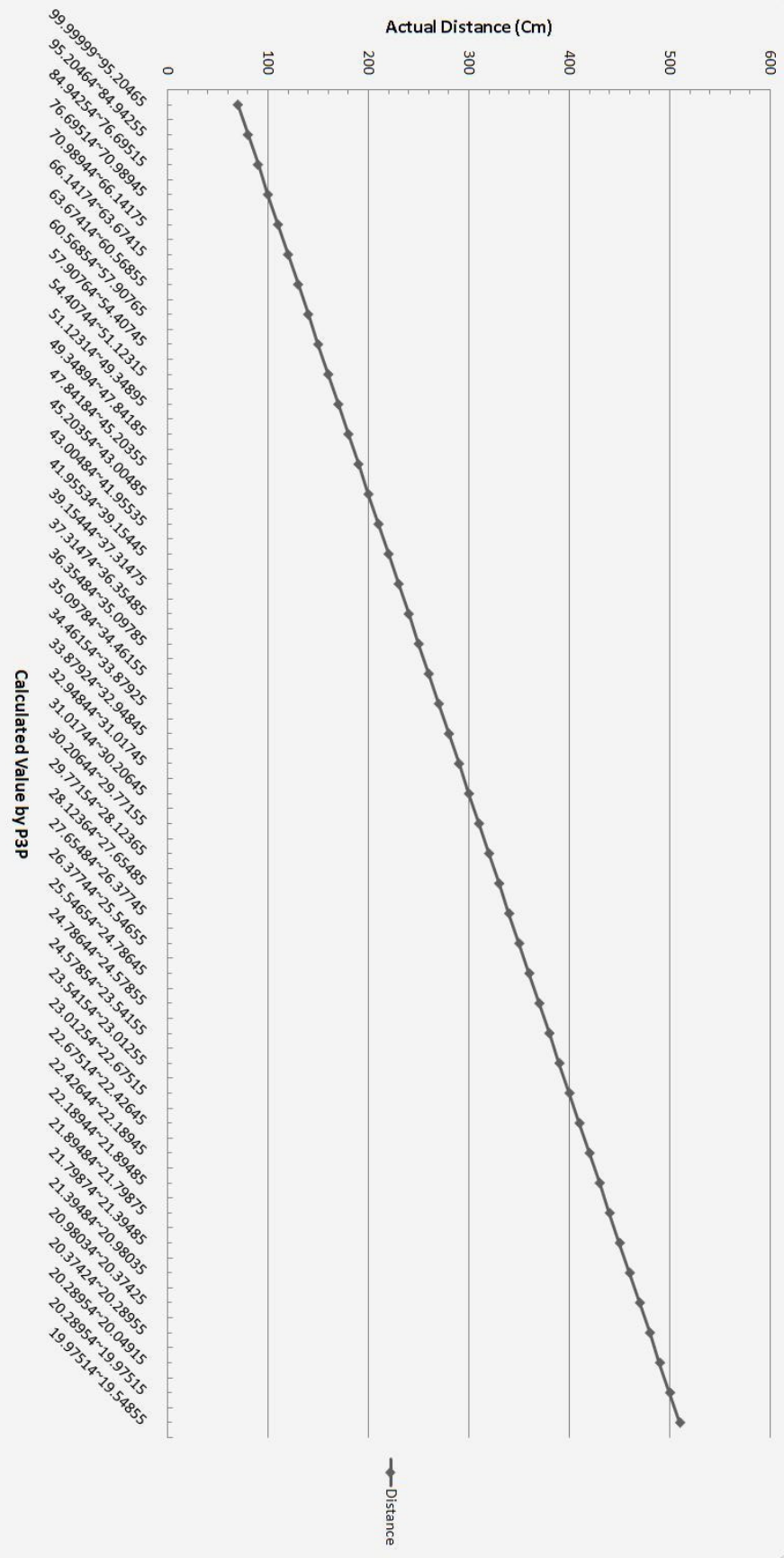


FIG. 4.4. *Actual distance calculated from measured relative distance*

TABLE 4.2
IR-LED Visibility Experiments

MODEL NO.	Max Length	Visible Angle	Visibility
SI5315-H	500cm	±60°	Stable
OPE5685	490cm	±45°	Somewhat Unstable
OPE5194WK	520cm	±35°	Most Stable
TLN201	510cm	±20°	Unstable
EL-1KL5	450cm	±10°	Indiscriminable

can be directly drawn from the following equation 4.1

$$\tan\theta = \frac{d}{2h}$$
$$\theta = \arctan\frac{d}{2h} \tag{4.1}$$

Most of cameras has angle of view in $54° \sim 60°$. Since we used camera with angle of view in 60°, from the equation 4.1 we can solve ratio about $h : d = 1 : 1.08$. The actual value of d is 30 centimeter for our experiment.

Thus we can summarize the following:

- High angle of view camera can increase minimum measure distance.
- With narrow LED pattern interval, we can decrease actual distance h but practically meaningless.
- With wider LED pattern interval, we can increase actual distance but dependent on MSV size.

From the experiments, we can identify the vision based localization is effective within the range from 30 centimeters to 520 centimeters with our LogiTech CAM camera. The 30 centimeter lower bound is due to the 30 centimeter interval of LED triangle edges. The 520 centimeter upper bound is due to the visible sight capability of LogiTech CAM camera. Thus 520 centimeter would be a maximum distance of computer vision based localization. However it is still meaningful since we can achieve very high accuracy in localization with these cheap, low grade cameras. The other idea for more localization distance is to use cameras with higher resolution.

From our experiments, we identified the correlation between actual distance from camera to colleague MSV and size of LED triangles in camera view. The results can be translated into graphical form as shown in figure 4.3.

From figure 4.3 the result shows the fluctuation of results with more than 500 centimeters which makes localization unstable. For applications which require the error range of 20 centimeters, we can use the results to 520 centimeters. Since our aim is to keep localization errors within the range of 10 centimeters, we decided to discard results more than 400 centimeters.

4.4. Experimental Result. From our experiment in the previous subsections we will provide the final result of computer vision based localization in this subsection. Figure 4.4 shows graphical version of final result.

Apart from the results in previous section, this figure shows actual distance up to 500 centimeters. From figure 4.3 we can observe errors in calculated values of P3P for more than 500 centimeter distance. These errors is due to the resolution limit of CAM camera which is 640×480. Even a small noise can vary actual distance of ten centimeters in the distance more than 500 centimeters.

Thus we conclude the accurate localization by computer vision can be done in the range of 70 centimeters to 500 centimeters with our camera equipments. For the localization in more than 500 centimeters, localization based on MSV trajectory tracking will be effective. In addition, for the localization in more than 30 meters, localization based on RSSI will be effective 6. Of course, the location information can be broadcasted and be used by the members of MSN in order to build maps, to correct location errors and so on.

5. Mobility Trail based Localization.

5.1. Map Building. Map building is one of the core parts of medium distance localization as well as for other distances and areas. The result of localization must be presented on local map and therefore be

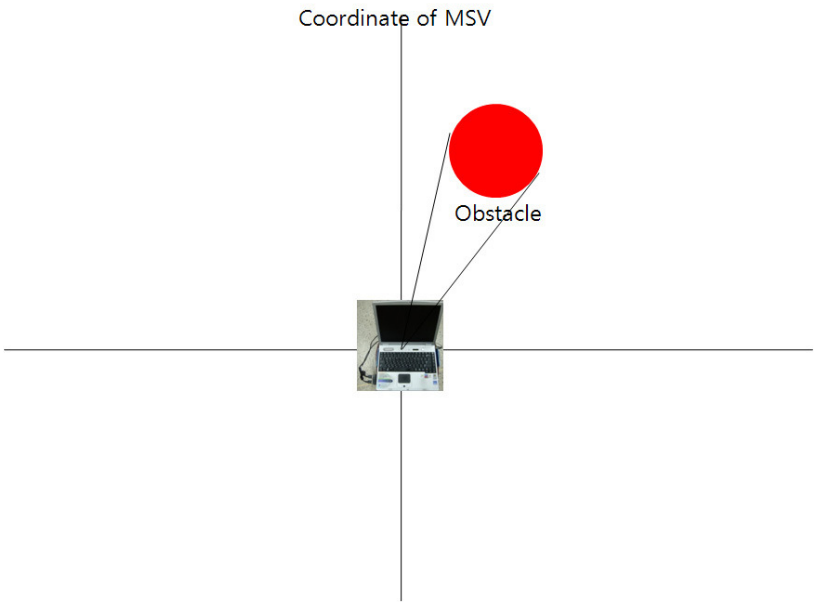


FIG. 5.1. *Relative Coordinate*

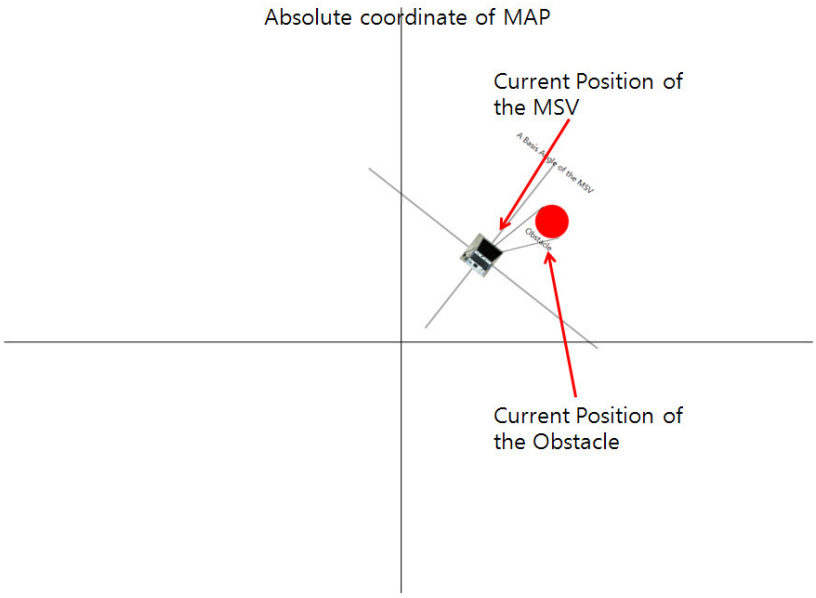


FIG. 5.2. *Absolute Coordinate*

transferred to global map. MSVs communicate with each other in order to combine local maps into global maps. The following information will be shown on a map.

- Untapped territory
- Territory with obstacle
- Territory with MSV
- Tapped territory
- Totally unknown territory

For map building we must consider relative coordinate and absolute coordinate. For example, obstacle information identified by MSV is in a form of relative coordinate. In relative coordinates, the very front of MSV is in angle 0 as shown in figure 5.1. This coordinate must be transformed into absolute coordinate as shown in figure 5.2 and therefore can be a part of map.

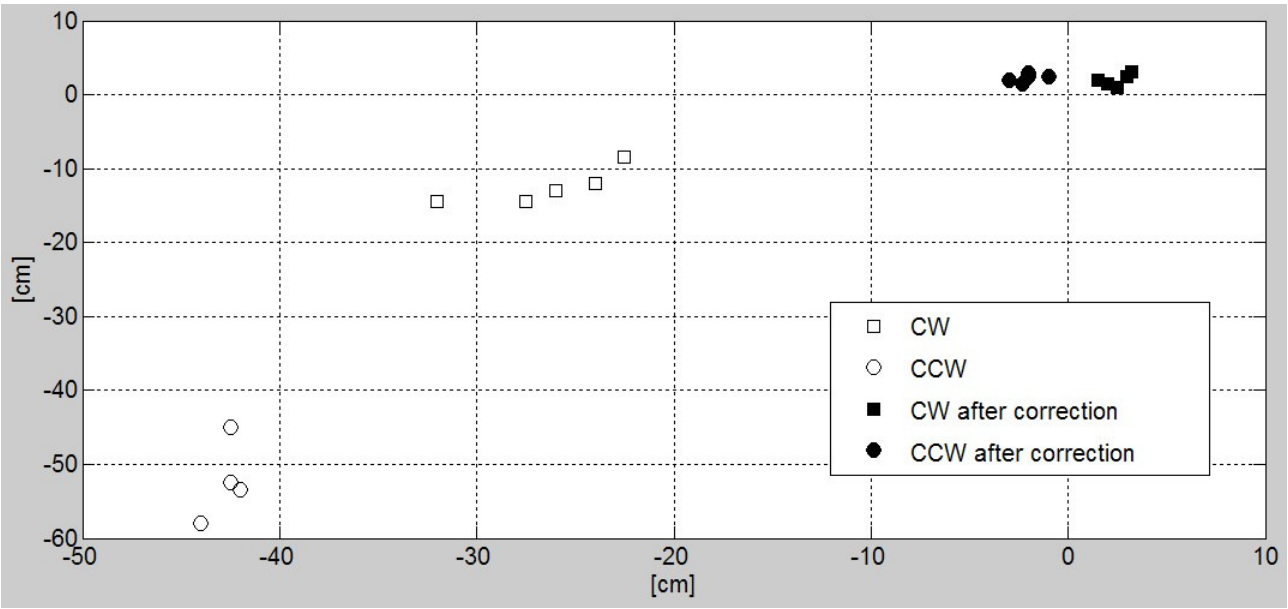


FIG. 5.3. *Result of UMBmark*

TABLE 5.1
COMPARISON BETWEEN GRID MAP AND TOPOLOGICAL MAP

MAPS	Grid MAP	Topological MAP
Advs.	Precise presentation of geography of environment Ease of algorithm design: environmental modeling, path finding, localization by map-matching	Simple presentation of environment and simple path planning Tolerance of low accuracy mobile sensors Natural interface to users
Disadvs.	Difficulty in path planning Requirement of large memory and computation Poor interface to symbolic problem solver	Impossibility of large map building with inaccurate, partial information Difficulties in map-matching: difficulties in calculation of pivot sensor value Difficulties in dealing complex environment

Local map is usually in a form of grid map. However in case of global map with huge capacities, grid map is very inefficient. Therefore we will use topological map for global map as presented by Kuipers and Bynn [6]. Thrun [8] presented a hybrid approach of both maps and we will consider it as our ultimate format of global map. Table 5.1 compares advantages and disadvantages of grid and topological map.

With mobility trajectory tracking, medium range localization can be implemented by use of local map. Each MSV moves autonomously and build its own local map. In the following subsection, we will discuss error corrections of mobility tracking based approach which is essential to guarantee the accuracy of localization.

5.2. Dead-Reckoning. For the medium distance localization, we decided to utilize mobility trail. We define the range of medium distance between 4 meters and 40 meters since our vision based short distance localization covers within the range of 5 meters and RSSI based long distance localization is effective outside the range of 30 meters. Our aim is to trail the mobility of MSV and to record the trail on the local map with reasonable accuracy for medium distance localization. Every driving mechanism for mobile sensors or even mobile robots has mechanical errors and it is impossible to avoid such errors practically. We can summarize the cause of driving errors as followings:

- The difference between the sizes of two (left and right) wheels
- The distortion of wheel radius, i. e. the distance between average radius and nominal radius
- The wheel misalignment

- The uncertainty about the effective wheelbase
- The restricted resolution of driving motors (usually step motors)

Usually those errors are cumulated and final result will be void without proper error correction technique. However, in order to cope with those location errors due to mechanical errors, a method of dead-reckoning have been widely used and we also adopt such technique as well. Dead-reckoning is a methodology that calculates the moving distance of two wheels of MSV and derives the relative location from the origin of MSV.

Among the various versions of Dead-reckoning techniques, we used UMBmark technique from University of Michigan [4]. UMBmark analyzes driving mechanism errors and minimized the effect of driving errors. UMBmark analyzes the result of MSV driving in a certain distance and compensates mechanical errors of MSV driving mechanism. The driving results of rectangular course, both in clockwise(CW) and counter-clockwise(CCW), and then analyzed.

Two error characteristics are classified in Rotation angle error and Wheel mismatch error. Rotational angle errors are for the difference between actual wheel sizes and theoretical design sizes of wheels. Due to rotational angle errors, CCW driving after CW driving shows larger errors as usual. For example, actual wheel size bigger than designed wheel size results in insufficient rotation at corners and then rotational angle errors are cumulated for the whole driving. The following equation summarizes the rotational angle error which is depicted in [4].

$$E_d = \frac{D_R}{D_L}$$

where D_R is diameter of left wheel and D_R is diameter of right wheel. In short, E_d is a ration between diameters of left wheel and right wheel.

Wheel mismatch errors are from wheelbase mismatch. This error causes skews in straight driving. With wheel mismatch error, the error characteristic of CW driving is opposite to CCW driving. The following equation summarizes the wheel size error which is depicted in [4].

$$E_b = \frac{90^\circ}{90^\circ - \alpha}$$

where α is a value of rotational angle error. E_b stands for a ration between ideal and practical errors in rotation, i. e. wheel base error.

Mechanical errors are systematical errors and therefore can be predicted and analyzed, while non-mechanical errors cannot be predicted because non-mechanical errors are due to the driving environment. Non-mechanical errors are classified as follows:

- Uneven driving floor or ground
- Unpredicted obstacle on driving course
- Slipping while driving

We applied UMBmark to our MSV and the following subsection shows the result.

5.3. Driving Error Correction of MSV. We composed a set of experiment for MSV driving in order to apply UMBmark. The driving experiments have been made on the flat and usual floor with the rectangular driving course of 4 × 4 meters. As shown in [4] both CW and CCW driving have been made and error values have been measured. These error values are incorporated in our software system and MPU controllers.

With the following equations from [4] we can find the error value for error correction.

$$b_{actual} = E_b \times b_{nominal}$$

where b_{actual} is an actual wheelbase and $b_{nominal}$ is a measured wheelbase.

$$\Delta U_{L,R} = c_{L,R} \times c_m \times N_{L,R}$$

Where U is actual driving distance, N is the number of pulses of the encoder, and c_m is the coefficient to convert pulse per centimeters.

Our experimental result with driving location correction by UMBmark dead-reckoning mechanism is shown in figure 5.3. Circled dotes are result from CCW driving and rectangular dotes are from CW driving. Empty

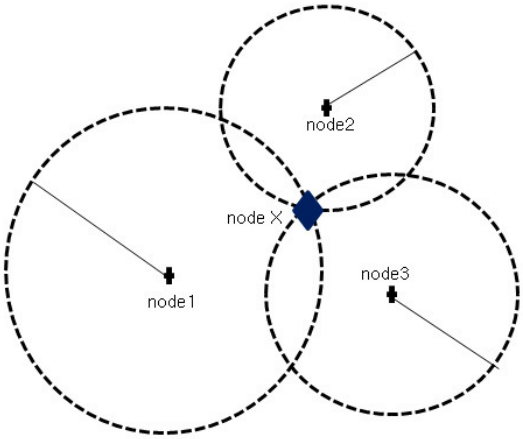


FIG. 5.4. *Triangulation With RSSI Measurement*

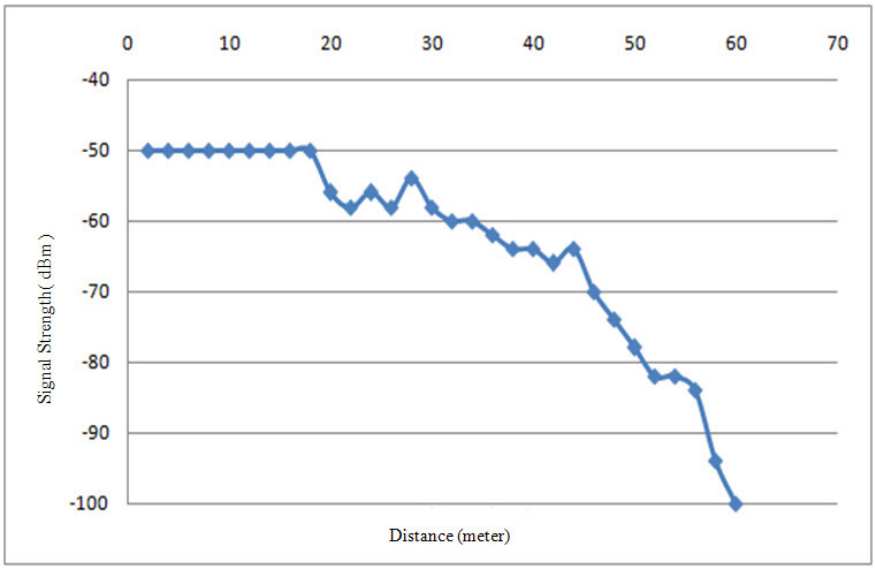


FIG. 5.5. *RSSI values According To Distance*

dotes are of uncorrected driving results while filled dotes are of driving results with UMBmark in 16 meter driving experiment. Note that the origin of MSV (starting point) at the coordinate (0,0) are at the upper right part of the figure. Without dead-reckoning technology, MSV returns to erroneous point than the origin point, at the left part of the figure. This MSV tends to show more errors with CW driving. With the application of UMBmark technique, we achieved faithful result within 10 centimeters of error range in total. Directional errors are within the range of 3 centimeters from the origin. Since our approach is for mechanical driving errors, non-mechanical errors can be avoided and thus we will introduce real-time correction of driving with the help of digital compass for the future researches. Thus it is possible to mention that the trail of MSV is in the correct location within the errors of 10cm in our experimental environments.

6. RSSI based Long Distance and Wide Area Localization. Our MSV are equipped with homogeneous 802.11 networking devices with RSSI facilities. With distance information we can do triangulation with at least three nodes and one anchor. Our monitoring station with monitoring program can act as an anchor. The 802.11 networking devices can be switched to AP (Access Point) mode so that each MSN can act as AP. With software modification that utilizes 802.11 device RSSI features, we can achieve RSSI based localization for our MSN.

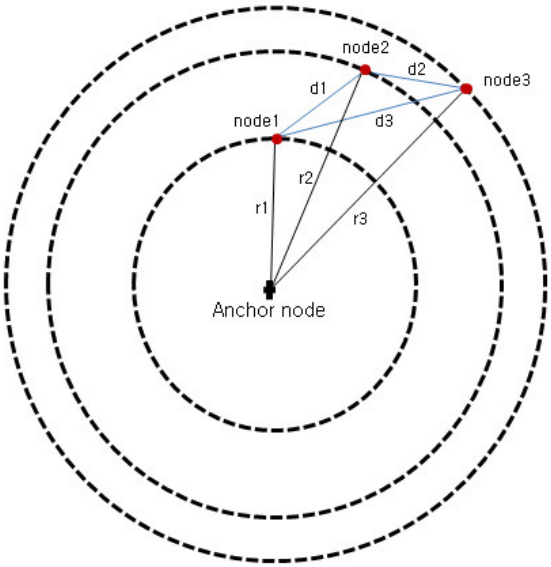


FIG. 5.6. *Relation Between One Fixed Anchor And Mobile Sensor Vehicles*

The unit of RSSI is in dBm (-50dBm $\sim$ -100dBm) and it designates distances between specified MSVs. As already mentioned, the RSSI value is very affective by environments, and we obtain error rate of 10 $\sim$ 15% in specific distance. The RSSI value is very sensitive with hardware vendor and the direction of AP [9]. Our experimental environment is as follows.

- Wide open area
- Intel Wireless LAN 2100 3B Mini PC Adapter
- WRAPI software model [11]
- One fixed anchor as monitoring station

For the calibration of our RSSI device, a set of experiment has been conducted and the results are shown in figure 5.5.

We can find within the distance of 20 meter, RSSI is no more useful for distance measure since the signal strength is too high. Our experiments shows RSSI based localization is useful more than 35m distance. The values are within error range of 15% by experiment. This is the main reason why we choose RSSI based localization for long distance, low accuracy localization.

From the distance information from RSSI sensing, we can do triangulation as shown in figure 5.4. For actual implementation, we have one fixed anchor and can do more precise localization with a known anchor coordinate as shown in figure 5.6. The figures show three mobile nodes one anchor node. The distance obtained from circle $r1, r2, r3$ can be obtained from RSSI values. Thus with this environment we can triangulate the coordinate node X from the intersection of circles drawn by node 1, node 2, and node 3 [12] [13].

Thus from the distance which can be obtained from RSSI values, let the distance be di from radius of circle ri The following algorithm 2 shows a procedure to find coordinates of each MSV with provided distance information by RSSI.

Figure 6.1 shows a final result in RSSI based localization. The x-axis stands for actual distance between MSVs and y-axis shows a distance calculated by algorithm 2. As we predicted the RSSI based localization is useful with the distance more than 30 meters. On the range where RSSI based localization is effective, we can see errors between actual distance and calculated distance. We believe it is tolerable since we have another method of localization with more accuracy within the distance of 30 meters. Of course, the distance information is not a sufficient condition for localization. The other information of direction of MSV can be obtained by digital compass on each MSV. Thus we implemented long distance, low accuracy localization.

Algorithm 2 Localization of Sensor Nodes with RSSI Measurement

```
Input : d1, d2, d3, r1, r2, r3
//Distance d1, d2, d3
//Circle r1, r2, r3
Output : SolutionList
LinkedList SolutionList
//Mobile Sensor Node Coordinates

for (each (x1,y1) on Circle r1)
{
 for (each (x2,y2) on Circle r2)
 {
  if (d1 ==
   distance between (x1,y1) and (x2,y2))
  {
   for(each (x3,y3) on Circle r3)
   {
    if (d2 ==
     distance between (x2,y2) and (x3,x3))
    {
     if (d3 ==
     distance between (x3,y3) and (x1,y1))
     {
      SolutionList =
       Coordinate (x1,y1),(x2,y2),(x3,y3)
}}}}}}
```

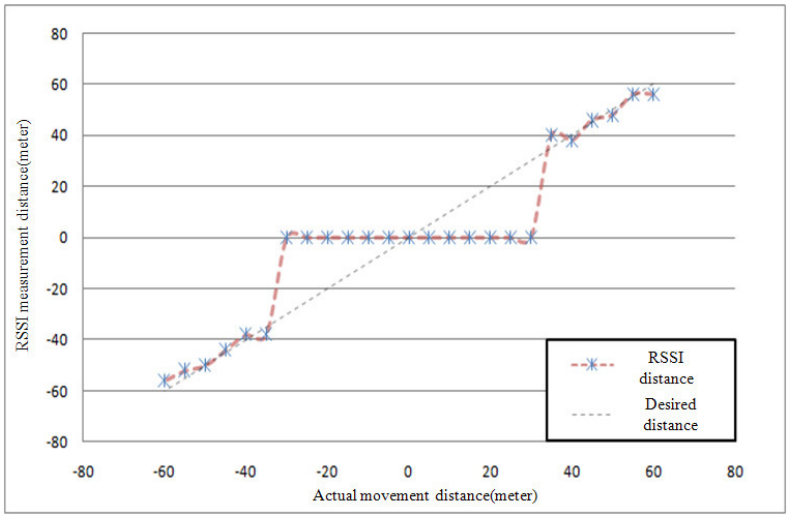


FIG. 6.1. *Actual and RSSI based Distance*

7. Conclusions. Of the localization methodologies for mobile sensor network, we combined three different categories of localization methodology. In addition for the experiment, we implemented mobile sensor vehicle as a node of mobile sensor network. We showed brief description of our mobile sensor vehicle including hardware and software functionalities. A computer vision based approach has been presented for the small area localization with a considerable range of preciseness. The driving mechanism hardware and software cooperate with each other and naturally achieve localization based on trajectory-tracking with the help of local map building, which is a medium distance and medium accuracy localization. The result of localization can be presented on local

maps and eventually be merger into global maps. In addition we showed RSSI based localization. The long distance, low accuracy localization can be implemented by commercial 802.11 networking devices only with software but without any other specific hardware device..

From these three levels of localization, we believe that we implemented useful localization system and will do more research using this platform. For example multiple MSV can cooperate and communicate each other and then a formation based on localization can be made. A smooth transition between these localization information for specific environment or application with probability model is our next goal to achieve.

Acknowledgments. The author gives special thanks to the past co-authors including Mr. Kwansik Cho, Mr. Jaeyoung Park and many other colleagues. Their contributions and allowances lead to this paper with united results.

REFERENCES

[1] J.-S. Gutmann, W. Burgard, D. Fox, K. Konolige, "An Experimental Comparion of Localization Method," IROS, 1998.
[2] J.-S. Gutmann, C. Schlege, AMOS, "Comparison of Scan Matching approaches for self-Localization in indoor Environments," EUROBOT, 1996.
[3] Lingxuan Hu and David Evans, "Localization for Mobile Sensor Networks", Tenth Annual International Conference on Mobile Computing and Networking (MobiCom 2004). Philadelphia, 26 September – 1 October 2004.
[4] J. Borenstein,L. Feng,Professors D. K. Wehe,Y. Koren,Grad Students,Z. Fan,B. Holt,B. Costanza ,UMBmark—A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots. 1995
[5] R.B.Fisher, A.P. Ashbrook, Robertson, C. and N. Werghi, A low-costrange finder using a visually located, structured light source. "3-D Digital Imaging and Modeling," 1999. Proceedings. Second International Conference on 4-8 Oct. 1999 Page(s): 24–33.
[6] B. Kuipers and Y.-T.Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations", Journal of Robotics and Autonomous Systems, 1991
[7] J. Ollero Gonzalez and A. Reina, "Map Building for a Mobile Robot equipped with a 2D Laser Rangefinder", Proc. of the IEEE Int. Conf. on Robotics and Automation, pp 1904-1909, 1994.
[8] Zhengyou Zhang , "A Flexible New Technique for Camera Calibration," IEEE Transactions on pattern analysis and machine intelligence, Vol.22, No.11, November 2000.
[9] Martin A. Fischler and Robert C. Bolles, SRI International. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Communications of the ACM, June 1981.
[10] J. Borenstein and Y. Koren, "Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance, IEEE TRAN". Robotics and Auto, Vol 7, No. 4, 1991.
[11] WRAPI, http://sysnet.ucsd.edu/pawn/wrapi
[12] Anastasia Katranidou, "Location-sensing using the IEEE 802.11 Infrastructure and the Peer-to-peer Paradigm for Mobile Computing Applications," Heraklion, February 2006.
[13] P. V. Bahl and Padmanabhan, "Radar: An In-Building RF-based User Location and Tracking System," in Proceedings of the Conference on Computer Communications (IEEE Infocom), (Tel Aviv, Israel), March 2000.
[14] IEEE Std. 802-11.1997, "IEEE Standard for Wireless LAN Medium Access Control(MAC) and Physical Layer(PHY) Specification", June 1997.
[15] Kohsei Mstsumoto, Jun Ota and Tamio Arai, "Multiple Camera Image Interface for Operating Mobile Robot", Proceedings of the 2003 IEEE lnternational Workshop on Robot and Human Interactive Communication Millbrae, California, USA, Oct. 31 – Nov.2, 2003
[16] Brenden Keyes, Robert Casey, Holly A. Yanco, Bruce A. Maxwell, Yavor Georgiev, "Camera Placement and Multi-Camera Fusion for Remote Robot Operation", IEEE Int'l Workshop on Safety, Security and Rescue Robotics, Gaithersburg, MD, August, 2006
[17] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, Pamela Mahoney, "Stanley: The Robot that Won the DARPA Grand Challenge". Journal of Field Robotics 23(9), 661–692 (2006)
[18] A. Elfes, "Occupancy grids: A Probabilistic Framework for Robot Perception and Navigation", PhD thesis, Department of Electrical and computer Engineering, Carnegie Mellon University, 1989
[19] Magnus Lindhe, Karl Henrik Johansson, "Communication-aware trajectory tracking", IEEE International Conference on Robotics and Automation, 2008, pp. 1519–1524, 2008, IEEE

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 325–336. http://www.scpe.org
ISSN 1895-1767

OPEN ENVIRONMENT FOR PROGRAMMING SMALL CONTROLLERS ACCORDING TO IEC 61131-3 STANDARD

DARIUSZ RZOŃCA, JAN SADOLEWSKI, ANDRZEJ STEC, ZBIGNIEW ŚWIDER, BARTOSZ TRYBUS, AND LESZEK TRYBUS*

Abstract. A control engineering environment called CPDev for programming small controllers in ST, FBD and IL languages of IEC 61131-3 standard is presented. The environment consists of a compiler, simulator and hardware configurer. It is open in the sense that: (1) code generated by the compiler can be executed by different processors, (2) low-level components of the controller runtime program are developed by hardware designers, (3) control programmers can define libraries with functions, function blocks and programs.

Of the three IEC languages, ST Structured Text is a basis for CPDev. FBD diagrams are translated to ST. IL compiler uses the same code generator. The runtime program has the form of virtual machine which executes universal code generated by the compiler. The machine is an ANSI C program with some platform-dependent components. The machines for AVR, ARM, MCS51 and x86 processors have been developed so far. Applications include two controllers for small DCS systems and PC equipped with I/O boards. CPDev may be downloaded from `http://cpdev.prz-rzeszow.pl/demo`.

Key words: control engineering tool, IEC 61131-3 standard, ST language compiler, multi-platform virtual processor

1. Introduction. Remarkable number of small-and-medium-scale companies in Europe manufacture transmitters, actuators, drives, PID and PLC controllers, and other control-and-measurement equipment. Engineering tools for programming such devices are often fairly simple and do not correspond to IEC 61131-3 standard [4], required by growing number of customers. The problem may be solved to some extent by developing open engineering environments for programming small control devices based on AVR, ARM, MCS51 or other microcontrollers according to IEC languages (61131-3 will be dropped for brevity). Development of such environment called CPDev (*Control Program Developer*) was initiated by the authors at the end of 2006.

The CPDev is open in the following sense:

- code generated by the compiler can be executed by different processors,
- low-level components of runtime program are provided by hardware designers,
- control programmers create their own libraries with reusable program units.

The CPDev compiler generates an intermediate, universal code executed by runtime interpreter at the controller side. Different processors require different interpreters. This resembles somewhat the concept of Java virtual machines [7] capable of executing programs on different platforms. Hence the interpreters of the CPDev universal code are also called *virtual machines*.

The same approach was adapted earlier in ISaGRAF package from ISC Triplex [5] (now in Rockwell). ISaGRAF universal code is called TIC (*Target Independent Code*) and may be executed on platforms supporting Windows, Linux, VxWorks, QNX and RTX. Much simpler CPDev does not impose such requirements, however. Another open environment called Beremiz [11] compiles IEC language code into C/C++ program, to be translated further into processor code. In this case commercial restrictions on the use of C/C++ compilers may matter sometimes.

This paper follows a few earlier publications, e.g. [9, 10], which reported on CPDev development. The content is organized as follows. For the reader not familiar with IEC standard, Sec. 2 provides some information on programming in high-level ST language. Components of CPDev, user interface, standard functions and libraries with function blocks are described in Sec. 3. Section 4 characterizes scanner, parser and code generator of ST compiler, written in C# at Ms .NET platform. Some instructions of the universal code called VMASM (*Virtual Machine Assembler*) are also presented. Section 5 describes operation and structure of the virtual machine. The machine is written in industry standard C and consists of universal and platform-dependent modules. Platform-dependent modules are written by hardware designers. Section 6 characterizes development of user function blocks, both in ST and C languages. Blocks written in C become components of the virtual machine. Programming in graphical FBD and textual IL languages is described in Sec. 7. FBD diagram is translated to ST and then compiled. Applications of CPDev for programming a small control-and-measurement

*Department of Computer and Control Engineering, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, 35-959 Rzeszow, ul. W. Pola 2, Poland, (`{drzonca, js, astec, swiderzb, btrybus, ltrybus}@prz-rzeszow.pl`).

distributed system, controllers of ship control-and-positioning system and a softcontroller based on PC with I/O boards are presented in Sec. 8.

2. A few notes on IEC 61131-3. The IEC 61131-3 standard [4] defines five programming languages, LD, IL, FBD, ST and SFC, allowing the user to choose the one suitable for particular application. Instruction list IL and Structured Text ST are text languages, whereas Ladder Diagram LD, Function Block Diagram FBD and Sequential Function Chart SFC are graphical ones (SFC is not an independent language, since it requires components written in the other languages). Relatively simple languages LD and IL are used for small applications. FBD, ST and SFC are appropriate for medium-scale and large applications. John and Tiegelkamp's book [6] is a good source to learn IEC programming.

ST is a high-level language originated from Pascal, especially suitable for complicated algorithms. Equivalent code for a program written in any of the other four languages can be developed in ST, but not vice versa. Hence most of engineering packages use ST as a default language for programming user function blocks. Due to such reasons, ST has been selected as a base language for the CPDev environment.

2.1. Data types. Data types, literals (constants) and variables are common components of IEC languages. Names (identifiers) are typical, although there is no distinction between capital and small characters. The standard defines twenty elementary data types, several of which are listed in Table 2.1 together with memory sizes and ranges (in CPDev). `BOOL`, `INT`, `REAL` and `TIME` are most common. `FALSE`, `13`, `-4.1415` and `T#1m25s` are examples of corresponding constants.

Table 2.1
Several elementary IEC data types

Type	Size (range)	Type	Size (range)
`BOOL`	1B (0, 1)	`LREAL`	8B IEEE-754 format
`BYTE`	1B (0 ... 255)	`TIME`	4B (-T#24d20h31m23s648ms ... T#24d20h31m23s647ms)
`WORD`	2B (0 ... 65535)		
`INT`	2B (−32768 ... 32767)	`TIME_OF_DAY`	4B (00:00:00.00 ... 23:59:59.99)
`REAL`	4B IEEE-754 format		

The standard defines three levels for accessing variables, `LOCAL`, `GLOBAL` and `ACCESS`. `LOCAL`s are available in the program, function block or function. `GLOBAL`s can be used in the whole project, but programs, function blocks or functions must declare them as `EXTERNAL`. `ACCESS` variables exchange data between different systems.

2.2. POU units. Programs, function blocks and functions, called jointly Program Organization Units (POUs), are components of IEC projects. Function blocks, designed for reuse in different parts of program, are of crucial importance. A block involves inputs, outputs and memory for data from previous executions. Therefore the blocks must be declared as instances. The IEC defines small set of standard blocks, such as flip-flops, edge detectors, timers and counters. Three of them are shown in Fig. 2.1.

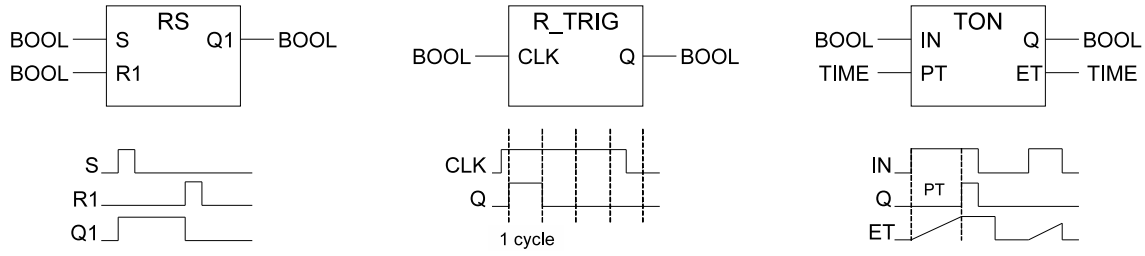


Fig. 2.1. *Examples of standard function blocks:* `RS` *flip-flop,* `R_TRIG` *rising edge detector,* `TON` *on-delay timer*

2.3. Programming. Programs written in ST or other languages begin with declarations of variables and instances of function blocks placed between `VAR_EXTERNAL` or `VAR` and `END_VAR` keywords. `GLOBAL` variables are declared before programs or separately. The declarations are followed by list of statements. The statements involve expressions which, when evaluated, yield results in one of defined data types, i. e. elementary (Table 2.1) or derived, such as alias, array or structure. The following operators are available (in descending priority): parenthesis, function evaluation, negation, power, arithmetic operators, Boolean operators.

ST language provides five types of statements:

- assignment := (Pascal symbol),
- selections IF, CASE,
- loops FOR, WHILE, REPEAT,
- early exits RETURN, EXIT,
- function and function block invocations.

Simple examples are presented in the following sections. Typical program looks like a sequence of function and function block invocations (calls).

3. CPDev environment. The CPDev consists of three programs executed by PC and one by the controller (Fig 3.1). The PC programs are as follows:

- CPDev compiler of ST language,
- CPSim simulator,
- CPCon configurer of hardware resources.

The programs have dedicated interfaces and exchange data through files in appropriate formats. The CPDev compiler (the same name as the package) generates universal code executed by virtual machine (VM) run by the controller. The VM operates as an interpreter. The universal code is a list of instructions of VM language called VMASM assembler. VMASM is not related to any particular processor, but close to typical assemblers. The compiler employs ST syntax rules, list of VMASM instructions and POUs from libraries. Besides the universal code the compiler generates some information for debugging and simulation by CPSim.

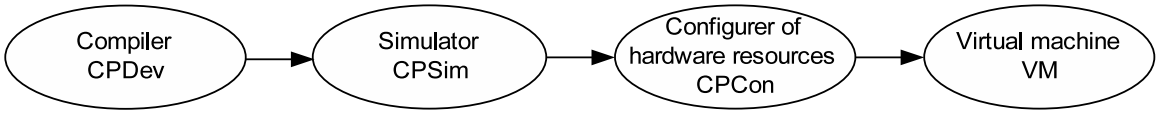


FIG. 3.1. *Components of CPDev environment*

Configuration of hardware resources by means of CPCon involves memory, input/output and communication interfaces. User specifications define memory types and sizes, numbers and types of I/Os and communication channels, validity flags, etc. Allocation of hardware resources has the form of a *map* that assigns symbolic addresses from ST programs to physical ones. By using it, the compiled code can be assembled for a particular platform to create final, universal executable code. From CPDev viewpoint, hardware platforms differ only in hardware allocation maps, whereas the compiled code is identical.

The CPDev environment has been recently extended by graphic editor of FBD diagrams and compiler of IL language. FBD diagram is automatically converted into ST code and compiled as above. Compilers of ST and IL differ in details only.

3.1. User interface. Main window of CPDev ST compiler is shown in Fig. 3.2. The window consists of three areas:

- tree of project structure, on the left,
- program in ST language, center,
- message list, bottom.

Frames of the areas can be adjusted and the contents scrolled.

Tree of the START_STOP project shown in the figure includes POU unit with the program PRG_START_STOP, five global variables from START to PUMP, task TSK_START_STOP, and two standard function blocks TON and TOF from IEC_61131 library. The program is written according to ST language rules. The first part involves declarations VAR_EXTERNAL of the use of global variables. Local declarations of the instances ON_DELAY and OFF_DELAY of the blocks TON, TOF are the second part. Program body consists of four statements. The first one turns a MOTOR on if START is pressed, provided that STOP or ALARM are not. Next three statements turn a PUMP on and off five seconds after the MOTOR (FBD diagram corresponding to this project is shown in Fig 7.1).

Global variables and the task are defined using separate windows (not shown). According to IEC standard the variables can be assigned CONSTANT and RETAIN attributes, and logical addresses. Task can be executed once, cyclically with a given period, or as soon as previous execution is completed. There is no limit on the number of programs assigned to a task.

Text of the project represented by the tree is kept in an XML file. Compilation is executed by calling Project->Build from the main menu. Messages appear in the lower area of the interface window. If there

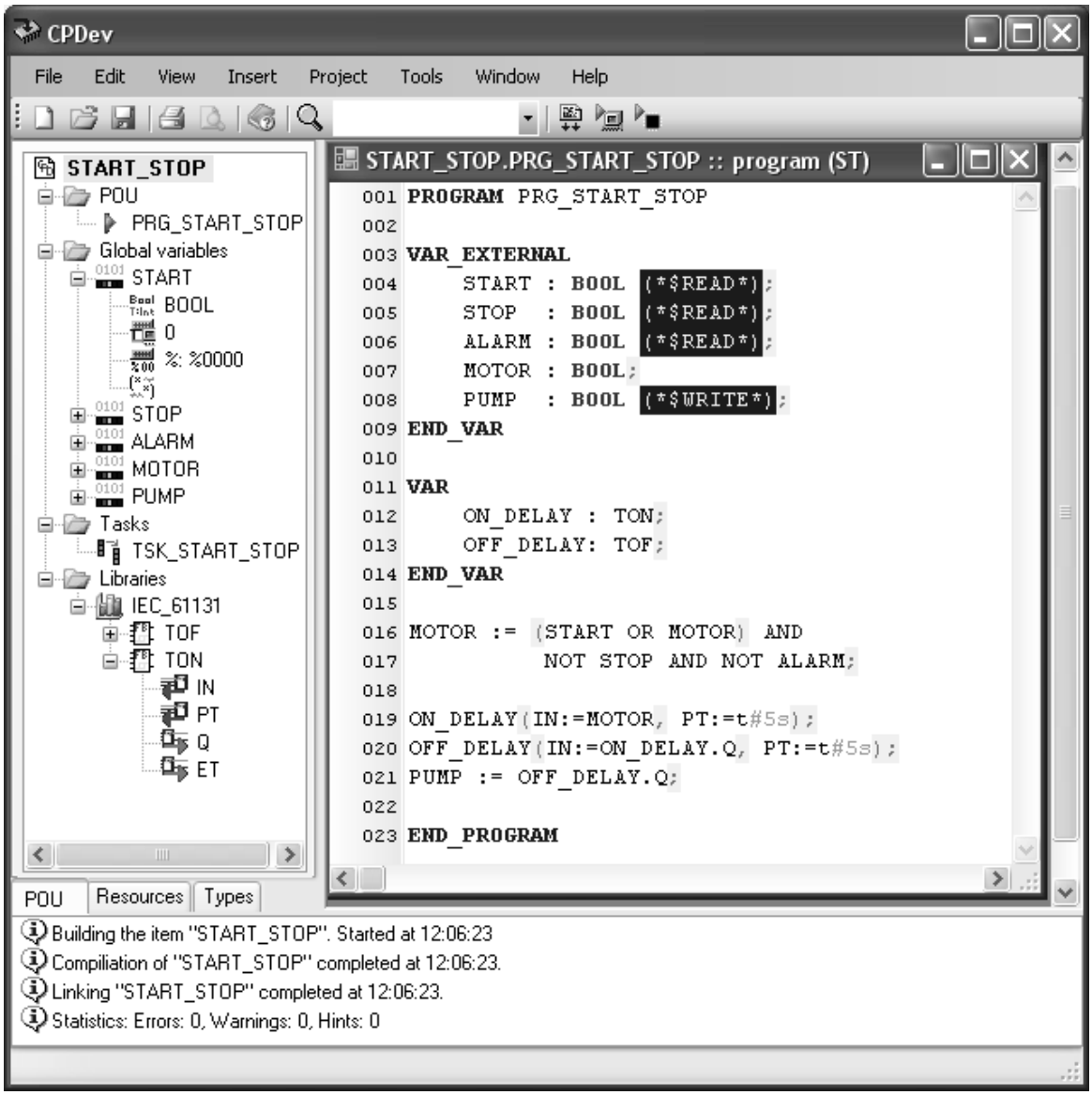


FIG. 3.2. *User interface of ST compiler (*`START_STOP` *project)*

are no mistakes, the compiled project is stored in two files. The first one contains universal executable code in binary format for the virtual machine. The second one stores mnemonic code, together with some information for simulator and hardware configurer (variable names, etc.).

3.2. Functions and function blocks. The CPDev compiler provides most of standard functions defined in IEC. Five groups of them followed by examples are listed below:

- type conversions: `INT_TO_REAL`, `TIME_TO_DINT`, `TRUNC`,
- numerical functions: `ADD`, `SUB`, `MUL`, `DIV`, `SQRT`, `ABS`, `LN`,
- Boolean and bit shift functions: `AND`, `OR`, `NOT`, `SHL`, `ROR`,
- selection and comparison functions: `SEL`, `MAX`, `LIMIT`, `MUX`, `GE`, `EQ`, `LT`,
- functions of time data types: `ADD`, `SUB`, `MUL`, `DIV` (IEC uses the same names as for numerical functions).

Selector `SEL`, limiter `LIMIT` and multiplexer `MUX` from selection and comparision group are particularly useful. Variables of any numerical type, i. e. `INT`, `DINT`, `REAL` and `LREAL` are arguments in most of relevant functions.

Two libraries of function block are available, namely:

- `IEC_61131` standard library,

- `Basic_blocks` library with simple blocks supplementing the standard.

The first one involves: (1) flip-flops and semaphore `RS`, `SR`, `SEMA`, (2) rising and falling edge detectors `R_TRIG`, `F_TRIG`, (3) up, down, up-down counters `CTU`, `CTD`, `CTUD`, (4) pulse, on-delay, off-delay timers `TP`, `TON`, `TOF`. Blocks typical for small multifunction controllers are in the second library, i. e. integrator, filters, max/min over time, memories, time measurement, etc.

4. ST language compiler. The task of the compiler is to convert XML source file with the project in ST language into a file with universal code in binary format. General diagram of the compiler operation involving scanner, parser and code generator is shown in Fig. 4.1.

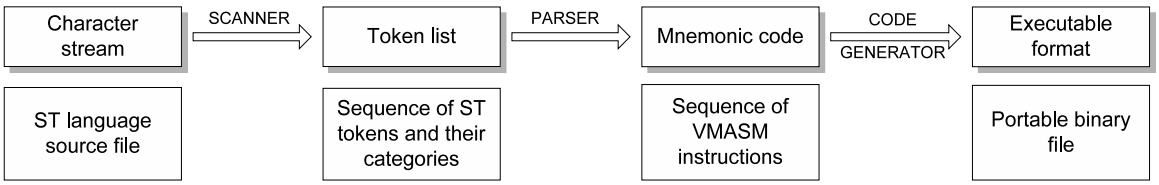


FIG. 4.1. *ST compiler components*

4.1. Scanner, parser and code generator. The scanner (lexical analyser) analyses character stream from ST source file and decomposes it into lexical units, i. e. tokens. The tokens are classified into categories such as identifiers, keywords, operators, constants (a few categories), delimiters, directives, comments, white spaces and invalid characters. The tokens with categories are collected on a list passed to the parser.

The parser operates according to top-down scheme with syntax directed translation [3]. By employing the ST syntax the parser recognizes consecutive token constructions from the scanner list. White spaces and comments are dropped. When correct construction is recognized the parser replaces it by a set of mnemonic instructions of the VMASM assembler. To do so, the parser employs built in elementary data types (Table 2.1) and list of VMASM instructions. Examples of these instructions are presented in Table 4.1.

TABLE 4.1
Examples of VMASM assembler instructions

Instruction	Meaning	Instruction	Meaning
`MCD`	Constant initialization	`GE`	Greater or equal
`MEMCP`	Assignment	`SHL`	Bit shift to the left
`ADD`	Addition	`JMP`	Unconditional jump
`SUB`	Subtraction	`JZ`	Conditional jump
`AND`	Logic product	`MEMCP`	Memory copy
`NOT`	Negation	`RETURN`	Return from function

Normally a single ST statement is translated into several VMASM instructions. Some translations require introduction of auxiliary variables and labels. Derived data types and POUs from libraries (functions, function blocks and programs) are also parsed. The mnemonic code is written in a special text format. The code can be consolidated with other mnemonic codes.

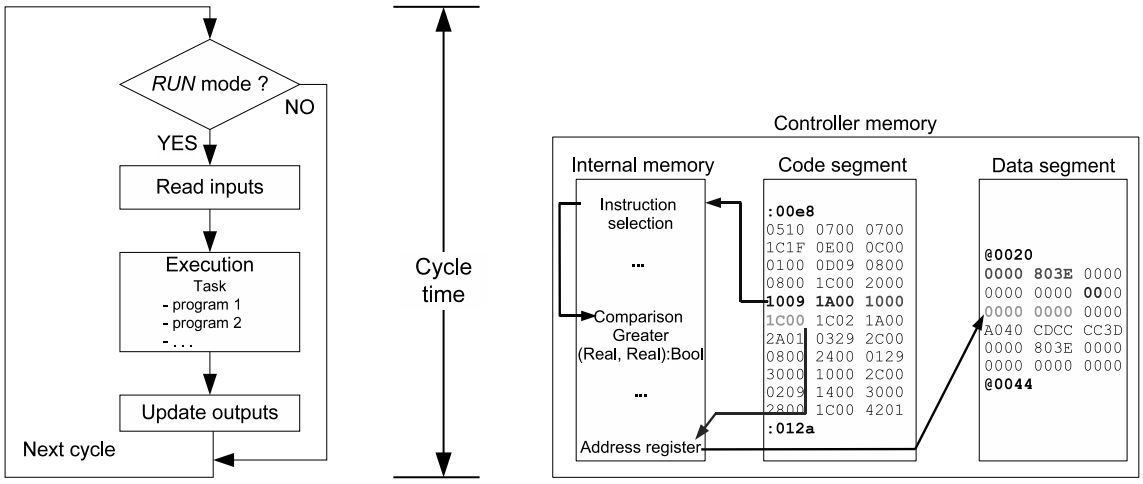
In the third step the code generator converts the consolidated mnemonic code into universal executable code in binary format. Mnemonics of the VMASM instructions, names of the variables and labels are replaced by corresponding number identifiers. To do so, the generator employs a *Library Configuration File* (LCF) with the identifiers of the instructions, numbers and types of the operands, and information how the operands are acquired (operand identifier may be an index to variable or a direct value). Each implementation of virtual machine is defined by specific LCF configuration file. Besides binary file with the executable code the compiler generates a text file with mnemonic code, some additional information for CPSim simulator and CPCon configurer (variable names, etc.) and compilation report (HTML).

4.2. Parser and code generator classes. Essential components of the compiler are designed as classes in C# language [1, 2]. Each token of ST language is encapsulated into an object of corresponding class. The classes inherit from an abstract `STIdentificator` class. During compilation, identifiers are collected into lists. The lists employ predicates for finding appropriate identifiers, what eliminates the need for hash tables. There is a list of global identifiers and local lists which store identifiers of functions, function blocks, programs, etc.

Identifiers in a list are checked for uniqueness. When identical names are found compilation is stopped and error reported. If local identifier hides a global one, the compiler produces a warning.

The parser generates text sequence of VMASM instructions for the code generator. Each instruction is represented by a mnemonic followed by operand names. Code generator replaces mnemonics and variable names with appropriate number identifiers (indexes). While processing an instruction, the generator extracts some information from libraries, e.g. operand size, type and passing method. The number identifier can be interpreted as a pointer to variable or as immediate value. Instructions resulting from compilation are represented by instances of `VMInstruction` class. The operand list `VMOperand` is also stored as a member of this class. By using lists of operands typical problems with fixed-size operand tables are avoided.

5. Multi-platform virtual machine. Binary file with the universal code and hardware allocation map from the CPCon configurer are downloaded into the controller, to be processed by virtual machine. Main features of the processing are characterized below.



a) Operation of VM b) Memory organization

FIG. 5.1. *Virtual Machine*

5.1. Operation cycle. Virtual machine is an automaton operating according to Fig. 5.1a. As indicated before, the machine is specific for a particular processor and works as an interpreter. The task consists of programs executed consecutively. The binary code involves number identifiers of the instructions and addresses of operands. The machine, similarly as a real processor, maintains program counter with the address of instruction to be executed, and base address of the data area with operands (specified for each POU). Given the instruction address, the machine fetches the identifier, decodes it, fetches the operands, and executes the instruction. Stack emulation and update of the base addresses permit multiple, concurrent calls of functions and function blocks. The machine monitors time cycle of the task and sets alarm flag if timeout appears. It also triggers input/output procedures responsible for external variables.

Allocation of software to memory segments is shown in Fig. 5.1b. The instructions and their operands are in the code segment (*read only*). Data segment contains global, local and auxiliary variables, some of them with constant values. The data segment can be accessed directly or indirectly by special virtual registers. The machine's internal memory keeps code of the interpreter, stacks and registers. There is no way of accessing internal memory from the program level. The machine is able to execute multiple instances of programs.

As shown in Fig. 5.2, the virtual machine consists of a few universal and platform-dependent modules to simplify implementation. The universal modules remain unchanged (if one neglects compilation of the source code for a given processor). The platform-dependent modules interface the machine to particular hardware, executing VM requests to low-level procedures. For instance, the module `Time&Clock` is associated with hardware, as it employs time interrupts to handle `TIME` data. `DATE_AND_TIME` data require real-time clock (RTC) on board. I/O functions provide interface to analog and binary inputs and outputs, and to communication fieldbus or network. The multitasking module is optional (not implemented yet), since it employs mechanisms of the host operating system.

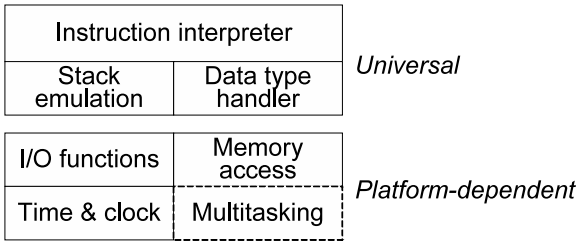


FIG. 5.2. *Universal and platform-dependent VM software modules*

The universal part of the virtual machine has been written in ANSI C, so it can be directly applied to different processors. As indicated in Sec. 4.1, the number of data types and the way in which the machine instructions are executed are defined by the LCF configuration file. For example, one can limit the number of elementary data types or define a subset of VMASM instruction to be used. A set of general specifications has been developed in CPDev for handling processor components (interrupt system, RTC) and external interfaces (I/O, communications). The specifications are in the form of prototypes of corresponding procedures (names, types of inputs and returned outputs). The prototypes do not depend on processor and hardware solutions.

The file with the prototypes is compiled together with the universal modules of the virtual machine. The contents (bodies) of the specification procedures can be prepared by hardware designers and, as a binary file, consolidated with the compiled universal modules. This gives the complete code of the virtual machine for given platform. Till now, the machines for AVR, ARM, MCS-51 and PC platforms have been developed.

We stress that the contents of low-level procedures dependent on hardware solutions may be written by designers themselves. This makes the CPDev package *open* in the hardware sense.

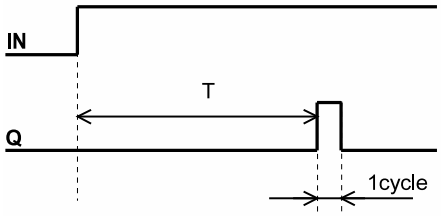
6. User defined function blocks. The CPDev environment allows the user to define function blocks both at PC side and at controller side, i. e. as components of virtual machine. The PC side blocks are written in ST, whereas the VM side ones are in C. However, the C blocks are still invoked in the main ST program compiled and downloaded from PC. So, as far as invocations are concerned, there is no difference between ST and C blocks.

6.1. ST blocks. User libraries are created in CPDev as typical projects which may include all kinds of POU units of IEC standard, i. e. programs, functions and function blocks. Declarations `VAR_INPUT` and `VAR_OUTPUT` determine input/output structure of functions and function blocks. There is no difference between programming of a project directly for controller implementation and programming a library. However, the library project is semi-compiled to VMASM mnemonics and not to binary form. So the last component of ST compiler, code generator (Fig. 4.1), is not needed. The file with mnemonics becomes user-defined library and is exported to *Libraries* folder.

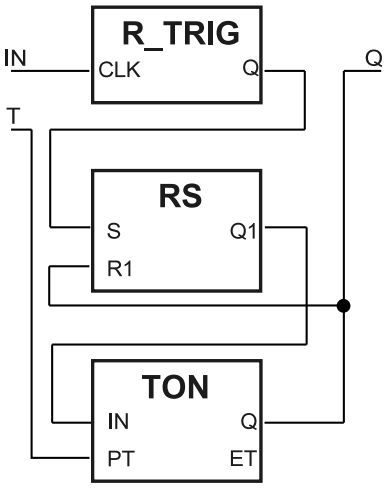
Example of user function block `FB_PULSE` is shown in Fig. 6.1. The block generates single pulse at the output `Q` after time `T`, since rising edge has appeared at the input `IN`. The program of the block may implement FBD diagram of Fig. 6.1b, with standard blocks `R_TRIG`, `RS` and `TON` from CPDev `IEC_61131` library (Fig. 2.1). Corresponding ST code is shown in Fig. 6.1c, with `FB_PULSE` belonging to the project `PROJ_MY_BLOCK` (top of Fig. 6.1c). XML file with `PROJ_MY_BLOCKS` source code should be saved for future extensions and modifications. Semi-compilation of the project yields a file with VMASM mnemonics, called, for instance, `My_Library`. This file must be exported to *Libraries*. If `FB_PULSE` is needed in a new project, both `My_Library` and `IEC_61131` must be imported (the latter to support the former).

6.2. C-language blocks. Such blocks are needed at hardware level to handle I/O and communication channels. Inputs and outputs are declared in ST, but the block body is implemented in C, at virtual machine side (declarations are also repeated). Directive `(*$HARDWARE_BODY_CALL...*)` informs CPDev compiler that the block is a component of VM.

Table 6.1 presents initial parts of the code of `GPS_GGA` block which provides serial communication with a GPS device according to NMEA protocol (GGA is a command in NMEA). Identifier `ID:0003` in the `(*$HARDWARE...)` directive means that `GPS_GGA` is the third of C language blocks at VM side. `Align:4` tells the compiler to locate the variables at addresses divided by 4.



a) Time diagram



b) FBD realization

```
PROJ_MY_BLOCKS.FB_PULSE :: function
001 FUNCTION_BLOCK FB_PULSE
002
003 VAR_INPUT
004    IN : BOOL;
005    T  : TIME;
006 END_VAR
007
008 VAR_OUTPUT
009   Q    : BOOL;
010 END_VAR
011
012 VAR
013   TRIG_B: R_TRIG;    RS_B: RS;
014   TON_B: TON;
015 END_VAR
016
017 TRIG_B(CLK:=IN);
018 RS_B(S:=TRIG_B.Q, R1:=Q);
019 TON_B(IN:=RS_B.Q1, PT:=T);
020 Q:=TON_B.Q;
021
022 END_FUNCTION_BLOCK
```

c) ST code

Fig. 6.1. *Function block* FB_PULSE

Table 6.1
Declaration of C language block for GPS interfacing

ST declaration	C declaration in VM
FUNCTION_BLOCK GPS_GGA (*$HARDWARE_BODY_CALL ID:0003; Align:4 *) VAR_INPUT PORT : BYTE; END_VAR VAR_OUTPUT UTC : TIME_OF_DAY; LAT : LREAL; LON : LREAL; ALT : LREAL; QUALITY : BYTE; END_VAR END_FUNCTION_BLOCK	typedef struct __declspec(align(4)) tagIO_GPS_GGA { /*inputs*/ VM_BYTE Port; /*outputs*/ VM_TIME_OF_DAY Utc; VM_LREAL Lat; VM_LREAL Lon; VM_LREAL Alt; VM_BYTE Quality; } IO_GPS_GGA, *PIO_GPS_GGA;
Structure of the body	
switch(ID) {... case 0x0003: { PIO_GPS_GGA arg = (PIO_GPS_GGA)GET_PARAM_POINTER();...}...}	

The block's PORT input specifies communication channel. The outputs determine UTC time, LATitude, LONgitude and ALTitude of actual position, together with QUALITY of GPS reading. We stress that besides the declarations there is no body in ST component of the block.

Structure tagIO_GPS_GGA defined at VM side repeats ST declarations with alignment, specifies type name and pointer type. Executions of C blocks are implemented by switch(ID) statement with bodies entered at successive cases. So the body of GPS_GGA is entered at case 0x0003. Function GET_PARAM_POINTER() returns pointer to the structure determined for the blocks instance in declaration VAR ... END_VAR in the main ST program. The pointer is of general type void*, so must be converted to the type PIO_GPS_GGA. The resulting

pointer is saved in `arg` variable, sufficient for further processing. Other C language blocks are implemented in the same way. Given such template, hardware designers can prepare C blocks themselves.

7. FBD and IL compiler. The CPDev environment has been extended recently with simple graphic editor of FBD diagrams and compiler of IL textual language, mainly for teaching purposes. ST compiler remains basic platform of the environment.

7.1. Programming in FBD. The graphic editor, called Blockers (Fig. 7.1), provides basic editing functions, i. e. inserting blocks into diagram, connecting inputs and outputs of the blocks, selecting and removing objects, zooming, etc. The blocks are chosen from CPDev libraries. Global input/output variables and constant values are also placed in the diagram. Built-in syntax checker verifies correctness. Resulting FBD diagram is saved in XML text file whose structure follows recommendations of PLCopen [13]. The XML file is then converted into ST language by means of `FBD2CPDev` translator. Connections between the blocks and instances of the blocks are represented by automatically created local variables of corresponding types. Convention of variable names is based on types of blocks in the diagram and on execution order.

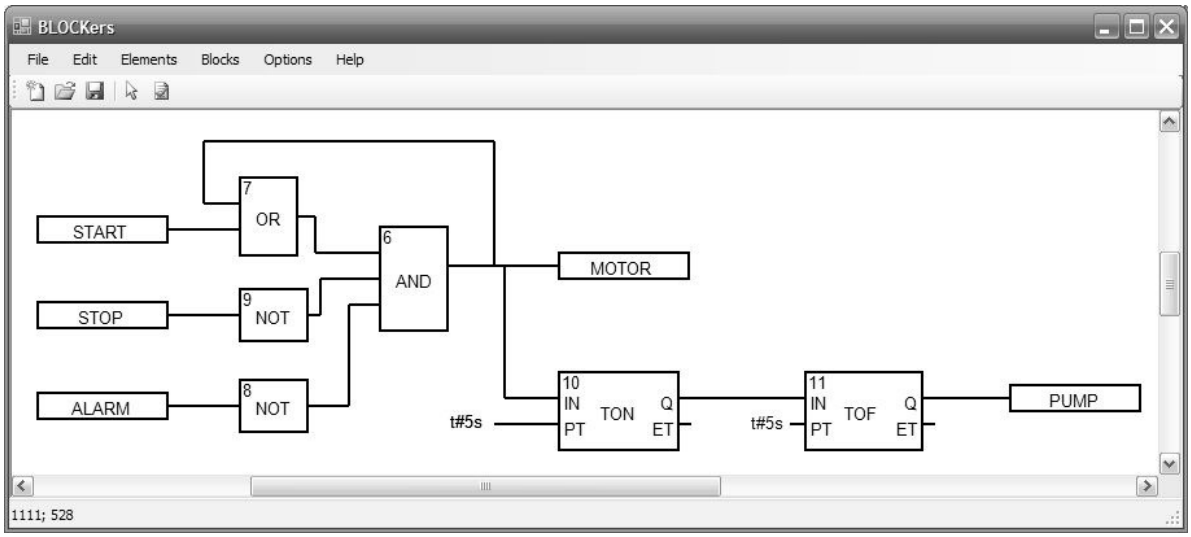


Fig. 7.1. *FBD diagram of the* `START_STOP` *system*

Figure 7.1 shows FBD diagram of the `START_STOP` system drawn using the Blockers editor. Numbers in the upper left corners of the blocks indicate execution order. Notice that in case of the function blocks `TON`, `TOF` the numbers may be used to distinguish instances. The variables placed in narrow rectangles on the left and right are interpreted as global. Equivalent ST code generated by `FBD2CPDev` translator is shown in Table 7.1 (compare Fig. 3.2).

Table 7.1
ST program converted from FBD

```
PROGRAM START_STOP

VAR_EXTERNAL
START : BOOL;
STOP : BOOL;
ALARM : BOOL;
MOTOR : BOOL;
PUMP : BOOL;
END_VAR

VAR
var_OR7_0 :  BOOL;
var_NOT9_0 :  BOOL;
var_NOT8_0 :  BOOL;
var_AND6_0 :  BOOL;
TON10 :  TON;
TOF11 :  TOF;
END_VAR

var_AND6_0 := AND(var_OR7_0,var_NOT9_0,var_NOT8_0);
var_OR7_0 := OR(var_AND6_0,START);
var_NOT8_0 := NOT(ALARM);
var_NOT9_0 := NOT(STOP);
TON10(IN := var_AND6_0, PT := t#5s);
TOF11(IN := TON10.Q, PT := t#5s);
MOTOR := var_AND6_0;
PUMP := TOF11.Q;

END_PROGRAM
```

It is seen that:

- connections between the blocks are represented by local variables `var_OR7_0` to `var_AND6_0`; name of a variable indicates source block of that variable,
- two instances `TON10`, `TOF11` are created, with names involving the block type and execution order.

Outputs of the instances, i. e. `TON10.Q` and `TOF11.Q`, are denoted in the standard way (compare Fig. 3.2).

7.2. Programming in IL. Since declaration parts of programs written in ST and IL are the same, and outcome of each compilation is a file with VMASM code, the compiler of IL language has been developed by extending the original ST compiler. The ST compiler generates the VMASM code from expression trees built of tokens acquired from ST code. By analysing a sequence of IL instructions one can create similar trees and employ them in succesive stages of compilation, in the same way as while compiling ST. This gives more efficient VMASM code than direct translation of IL instructions into VMASM, since VMASM, unlike IL, does not rely on the notion of accumulator. Accumulator is not needed in expression trees, typical for high-level languages.

Table 7.2
IL program for `START_STOP` *project*

```
PROGRAM PRG_START_STOP
                                    LD START
VAR_EXTERNAL                        OR MOTOR
    START : BOOL;                   ANDN STOP
    STOP  : BOOL;                   ANDN ALARM
    ALARM : BOOL;                   ST MOTOR
    MOTOR : BOOL;
    PUMP  : BOOL;                   CAL ON_DELAY(IN:=MOTOR, PT:=t#5s)
END_VAR                             CAL OFF_DELAY(IN:=ON_DELAY.Q, PT:=t#5s)

VAR                                 LD OFF_DELAY.Q
    ON_DELAY : TON;                 ST PUMP
    OFF_DELAY: TOF;
END_VAR                             END_PROGRAM
```

The `PRG_START_STOP` program of Fig 3.2 is rewritten in IL in Table 7.2. The instruction `LD START` loads `CR` register (*Current Result*; accumulator in IEC) with the value of `START`. Next the `CR` is `OR`ed with `MOTOR`, with the result in `CR`. The following `ANDN` negates `STOP`, `AND`s it with CR, always with the result in `CR`. Similarly for another ANDN. `ST MOTOR` saves `CR` in the variable `MOTOR`. `CAL` instructions invoke function blocks.

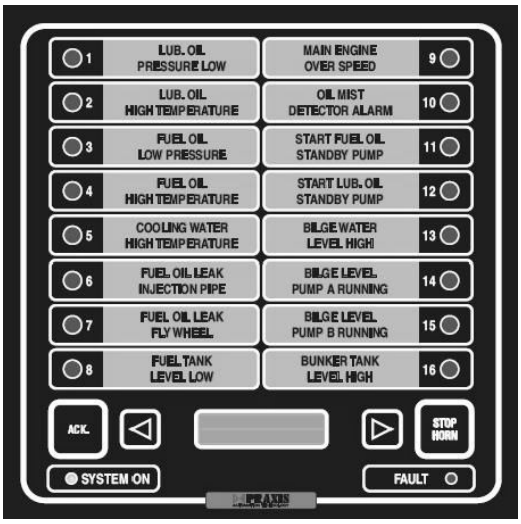
8. CPDev applications. The CPDev package is currently applied for programming new SMC controller from LUMEL, Zielona Góra, Poland. SMC operates as a central unit in small DCS systems involving distributed I/O modules, intelligent transmitters, PID controllers, etc. [12]. Development of another application in forthcoming version of MINI-GUARD Ship Control & Positioning System from Praxis Automation Technology, Leiden, The Netherlands, is in progress [8]. For lab and teaching applications PC-based softcontrollers can be used.

8.1. SMC controller. The SMC shown in Fig. 8.1a is based on Atmel AVR 8-bit microcontroller. Platform-dependent modules of virtual machine, i. e. interrupts, RTC and communication interfaces, have been written by LUMEL engineers, and sent to the authors in binary format. Consolidation of universal and LUMEL modules has resulted in a VM-SMC machine which, as SMC firmware, executes ST program compiled and downloaded from PC. The controller is equipped with two serial ports, one (*master*) for distributed I/Os and field devices, another (*slave*) for host PC or HMI panel. Modbus RTU protocol is applied (up to 230.4 kbaud). Third `Complex_blocks` library to implement self-tuning PID control loops is provided.

8.2. MINI-GUARD controllers. The MINI-GUARD system consists of seven types of controllers (Fig. 8.1b) involving NXP ARM7 16/32-bit microcontrollers. The controllers have application dedicated faceplates. Virtual machine for Atmel ARM7 has been sent to Praxis A.T., to be adapted for the NXP ARM7. The software to handle C language blocks described in Sec 6.2 has been developed especially for MINI-GUARD. The controllers communicate over Ethernet, external devices are connected via universal serial interface or OPC.

8.3. Softcontrollers with NI and InTeCo boards. A PC equipped with I/O board and executing a control program is called softcontroller. Two such boards can be used so far, namely NI-DAQ USB 6008 from National Instruments and RT-DAC/USB from InTeCo, Cracow, Poland (Fig. 8.1c,d). A common interface

a) SMC controller



b) Alarm Panel of MINI-GUARD

c) NI-DAQ I/O board

d) RT-DAC I/O board

Fig. 8.1. *Applications of CPDev package*

`CPDev.CPCom.ICommDev` has been developed, with provision for other types. Softcontroller is configured in two steps. First a board is selected from menu and I/O channels defined. Then global variables of the project are linked to the channels. Binary channels become `BOOL`s and analog one `REAL`s. Softcontrollers can be connected into DCS system by means of Modbus TCP protocol.

9. Conclusions and future work. CPDev environment for programming small controllers in ST, FBD and IL languages of IEC 61131-3 standard has been presented. The environment is considered open because compiled code can be executed by different processors, low-level software components are provided by hardware designers, and control programmers can create their own libraries with reusable program units. The compiler produces universal executable code processed by runtime virtual machine operating as interpreter. The machine is an ANSI C program composed of universal and platform-dependent modules. The machines for AVR, ARM, MCS51 (core) and x86 processors have been developed so far. User function blocks can be programmed in ST and C. The ST blocks are kept in CPDev libraries, whereas C blocks become components of virtual machine. FBD diagram is translated to ST and then compiled. CPDev has been used for programming controllers in two small DCS systems and for PC-based softcontroller with I/O boards.

Future work on CPDev will be motivated primarily by needs of the users. Next version will include structured data types and global arrays, at least two-dimensional (local arrays are available now). Current

simple FBD editor should be upgraded to more professional level. Depending on ST statements the compiled code is longer or shorter, as in the expression `x1 AND x2` vs. function `AND(x1, x2)`. Templates indicating more efficient solutions are important for the users. Virtual machine for FPGA platform with simple multitasking mechanism is currently under development.

REFERENCES

[1] A. Appel, J. Palsberg, *Modern compiler implementation in Java*, Cambridge University Press, Second edition, (2002).
[2] C# Language Specification, `http://msdn2.microsoft.com/en-us/vcsharp/aa336809.aspx`, (2007).
[3] K. Cooper, L. Torczon, *Engineering a Compiler*, Morgan Kaufmann, San Francisco, (2003).
[4] IEC 61131-3 standard: Programmable Controllers—Part 3, Programming Languages, *IEC*, (2003).
[5] ISaGRAF User's Guide, *ICS Triplex Inc.*, (2005).
[6] K. H. John, M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems* Berlin—Heidelberg, Springer-Verlag, (2001).
[7] T. Lindholm, F. Yellim, *Java Virtual Machine Specification - Second Edition*, Java Software, Sun Microsystems Inc, (2004).
[8] Mini-Guard Ship System, *Praxis Automation Technology B. V.*, `http://www.praxis-automation.com`, (2009).
[9] D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus, Mini-DCS System Programming in IEC 61131-3 Structured Text, *Journal of Automation, Mobile Robotics & Intelligent Systems*, Vol. 2, No 3, (2008).
[10] D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus, Programming controllers in Structured Text language of IEC 61131-3 standard, *Journal of Applied Computer Science*, Vol. 16, No 1, (2008).
[11] E. Tisserant, L. Bessard, M. de Sousa, *An Open Source IEC 61131-3 Integrated Development Environment*, 5^{th} Int. Conf. Industrial Informatics, Piscataway, NJ, USA, (2007).
[12] SMC, *Lumel S.A.*, `http://www.lumel.com.pl/en`, (2009).
[13] XML Formats for IEC 61131-3 ver. 1.01 Official Release, `http://www.plcopen.org`, (2007).

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2_ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.