13

Dynamic Control in Embedded Systems

Javier Vásquez-Morera[1], José L. Vásquez-Núñez[2]
and Carlos Manuel Travieso-González[3]
[1]*University of Costa Rica, Computer Science Deparment, San José,*
[2]*University of Costa Rica, Sede del Atlántico, Cartago,*
[3]*University of Las Palmas de Gran Canaria, Signals and Communications Department,*
Institute for Technological Development and Innovation in Communications, Campus
University of Tafira, Las Palmas de Gran Canaria,
[1,2]*Costa Rica*
[3]*Spain*

1. Introduction

Nowadays, different advances in technology have reduced the cost of micro-controllers and devices, such as sensors and actuators. In addition, the proliferation of Internet connections has led to the appearance of digital system controllers (DSC) that are designed to monitor and control equipment distributed by remote human intervention.

These DSC consists of a series of sensors controlled commonly by a local micro-controller, device capable of performing one or more tasks related to control (hardware - software) such as:

- View and control of physical variables.
- Schedule activities.
- Message service (e-mail, mobile phone).

While devices (DSC) may work in a distributed manner, it is usual to have the control intelligence permanently associated with one of the interconnected devices. In this work control means the attempt to impose predictability to unreliable entities that react to events. We also understand control policy as the predefined set of events that activate when control devices so dictate.

To carry out the distribution of control, we propose a distributed control model (DCM) that keeps separate control rules from control devices, hence any change in such policies do not necessarily alter the hardware or software used in a DSC. This method allows the incremental construction of longer useful life and more sophisticated control devices, whose flexible policies can be managed according to the user requirements.

It is the objective of this chapter to propose a strategy for the development of embedded systems, that incorporate aspects of computing interest as an abstraction of the behaviour and component-based design which allows embedded systems to have a number of features such as:

- Easy to scale up and adapt
- Easy to use
- Easy to maintain
- Reusable sensors and actuators

The structure of the chapter is divided in 4 sections:

The first section describes the Dynamic Control Model (DCM) as well as its main components, functionality and features.

The DCM in (Vásquez, 2010) permits the partial separation of the control in embedded systems allowing some of the rules (the basic ones) to be stored in the DSC and other rules in a central server which notifies each participant about the rules it manages and also the changes in the policies every time the user modifies any of them. Additionally, this section also explains the use of DCM in the development of monitoring and control applications that include embedded systems.

The second section explores in greater detail the issue of control rules managing the behaviour of an embedded system.

The third section deals with the use of software engineering techniques in the design of embedded systems. The Component Based Design (CBD) is introduced as a class reusing mechanism and the Contract-Based Design (CrBD) leads to the construction of robust components, easier to be reused than those created without CrBD.

The last section shows how to integrate the concepts mentioned above. The example proposed in this section serves as induction for those interested in the management of dynamics control rules.

2. Architecture of the Dynamic Distributed Control Model (DCM)

In the field of software engineering, the n-tier architecture is a term used to describe a methodology for designing and deploying applications. The methodology divides the functionality of an application into logical layers that interact to form a complete system.

The basic separation of an application proposes to divide the functionality of the system into three logical parts; the first one abstracts the interface (user services), the second one abstracts the rules of business services (business logic), and the third one functions as a data repository.

Nowadays, the development of distributed control systems is based on Internet technologies which include not only Web applications, but also the use of high-level languages, design and object-oriented programming, open protocols among others. Hence, the features and advantages of the methodology of n-layers are appropriate to the design and implementation of these systems.

Generally, in a DSC the control policies are set at the construction stage, which makes the device less flexible and not to respond adequately to changes in the requirements that users experience. The use of three-tier architecture allows the dynamic modification of composition and behaviour of both the abstract control policies and the DSC.

Figure 1 shows the 3-tier architecture used in the proposed model for distributed dynamic control (DCM). In the client layer one can find monitoring devices (Device Sensor) implemented through logic circuits. In the business layer the user finds Web services containing dynamic control rules that detect and process events. In the server layer is the database that can store data of users, devices, policies and events.

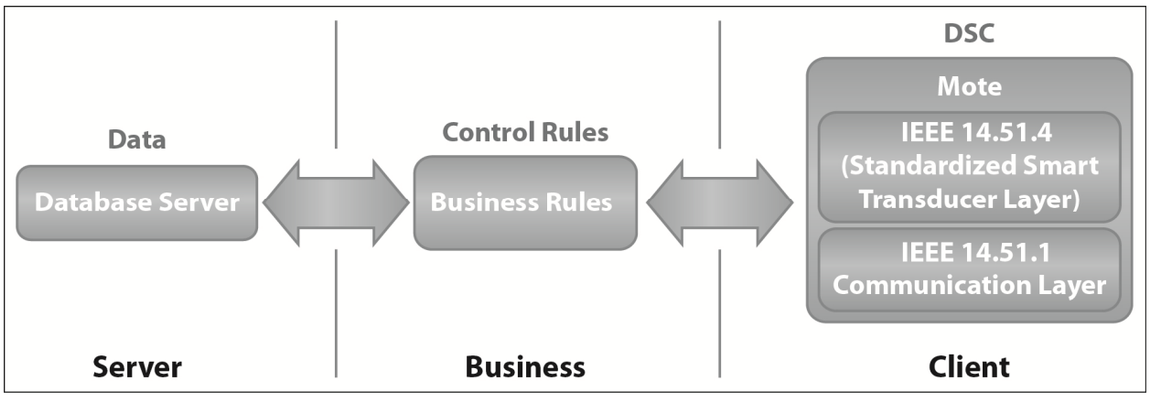


Fig. 1. Architecture for the control scheme with distributed intelligence.

2.1 Control device characterization

Figure 2 shows that control devices can be designed using the Model View Controller pattern, allowing abstract interface for managing devices. The distributed controller and the generic device model created by adding persistent components are grouped in 3 categories:

1. The sensors that conform the set of artefacts that are permanently recording the occurrence of events.
2. The actuators that are the set of artefacts reacting to determined rules, instructed to do so by a driver.
3. The microprocessors that can be microcontrollers or even personal computers.

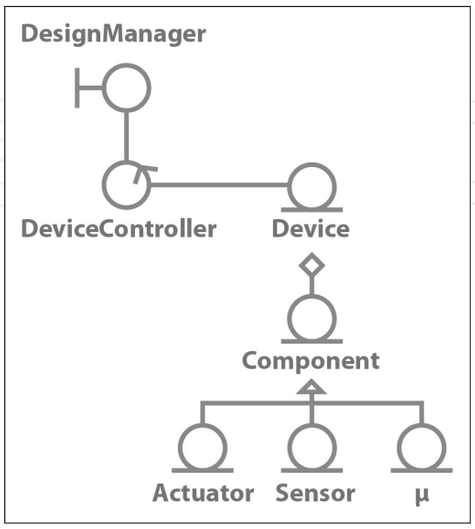


Fig. 2. Device design using reusable components.

From that abstract point of view, any device in the DCM is modelled as an object that has a number of attributes and that exhibits a behaviour which can be implemented in hardware or in software as shown in Figure 3. In this context, the components of a device are seen as the attributes of the same. Additionally, the behaviour of the object in the DCM is determined by its operations, and in the case of a device, the most common operations are related to the query and management of sensors and actuators, providing a service and setting its mode of operation.

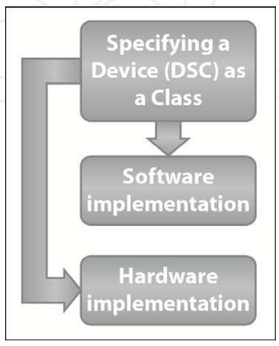


Fig. 3. Device implementation choice.

In designing a device as an object, encapsulation property allows hiding the construction details of the device. As a result, using this device requires only the knowledge of the set of operations or commands to interact with it. By using encapsulation in the device design, the control system obtains increased flexibility and better adaptability to the changes experienced by the devices as well as to changes in the user requirements. Figure 4 shows a specification of a DSC using a class diagram.

To integrate a device into the system, besides its activation, it is required to obtain participating control policies from a database. Once the device is operational, it can be accessed and managed remotely.

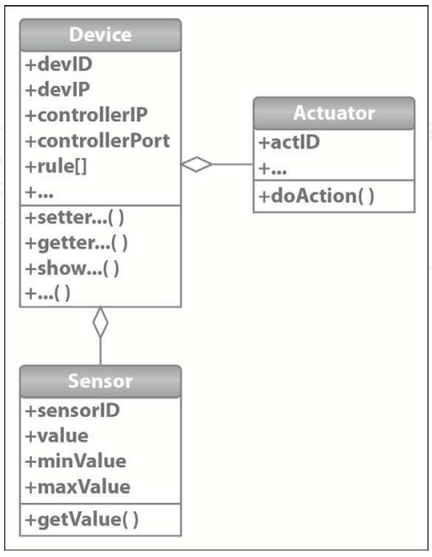


Fig. 4. DSC Diagram Class Specification.

2.2 DCM implementation using free software tools

Free software tools were used for the implementation of the DCM. The DSC device is composed by a general purpose microcontroller with an Ethernet interface built into the chip, making easier their connection to the network. Other components of the DSC are temperature sensors, humidity sensors, and electrical switches. The Events Controller is a Java application. The database, implemented with MySQL, contains the entities and relationships needed to show the performance characteristics of the model. The database was populated with data from users, devices, log of events and control rules. Finally, Tomcat was used as web application container, consisting of XHTML pages, EJB, JSP, and Servlets. This application provides the user a platform to search and specify control rules in the database, and to communicate directly with the DSC.

Once DCM is developed and implemented, there have been three important results:

1. Ability to alter the runtime behaviour of a DSC
2. Integration of open source technologies in the design and implementation of embedded systems
3. DCM_NIST model (Schneeman, 1999) was increased to enable distribution of dynamic control.

As Figure 5 shows, the use of DCM procures to obtain the following advantages:

- Adequacy, each manager can change the response that actuators must take after an event. This response is so flexible that in the future the manager can modify the actions.
- Automation, with this model it is possible to decide what sensing variables are needed to control in an automatic way.
- Scalability, the type and quantity of sensors and actuators can change in the future without discarding the old ones.
- Simplicity, the web interface allows interacting with the system in an easy way.
- Remote access, a user with an internet access point (mobile or not) may control the system and program the response of the actuators.

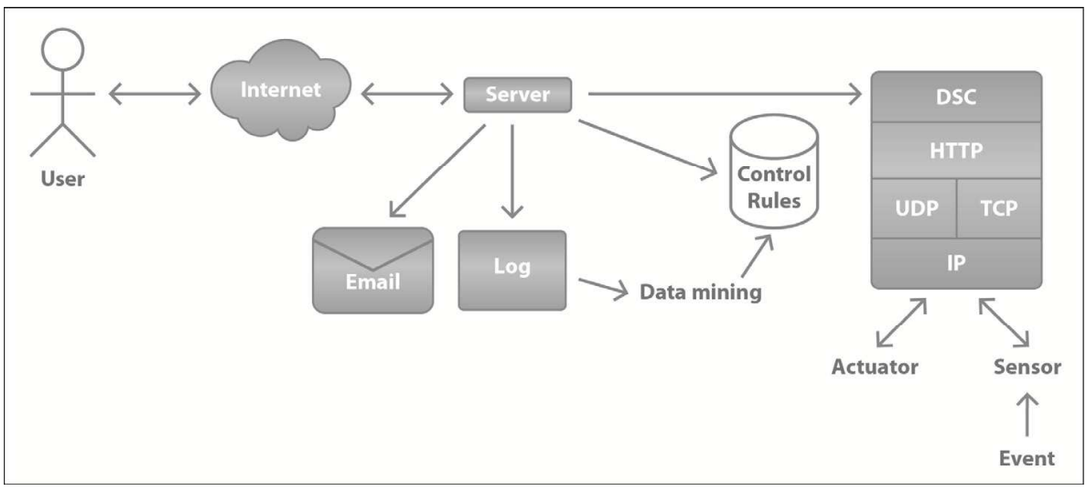


Fig. 5. DCM Implementation.

3. Management of control rules for the DCM

A control rule consists of one or more events and a series of actions that respond to the events. Any event generated by the sensors is recorded in a log, is evaluated against the control rules and if the necessary conditions are fulfilled, the controller executes the respective actions.

The MCD allows devices to record the occurrence of control events and take appropriate actions such as to activate some of the control actuators and communicate with other devices to delegate the monitoring of other rules and to enable its respective reaction, or notify by mail or messages to mobile devices.

As stated, the MCD provides three basic components:

- a set of sensors (e.g. 1,..,4)
- a set of dynamic rules of control ($Rule_a$ ... $Rule_p$)
- a set of actuators ($Actuator_1$ ... $Actuator_n$)

The microprocessor monitors the activation events from the sensors and communicates these events to the control rules engine, which is responsible for verifying those rules and triggers the respective actuators, see Figure 6.

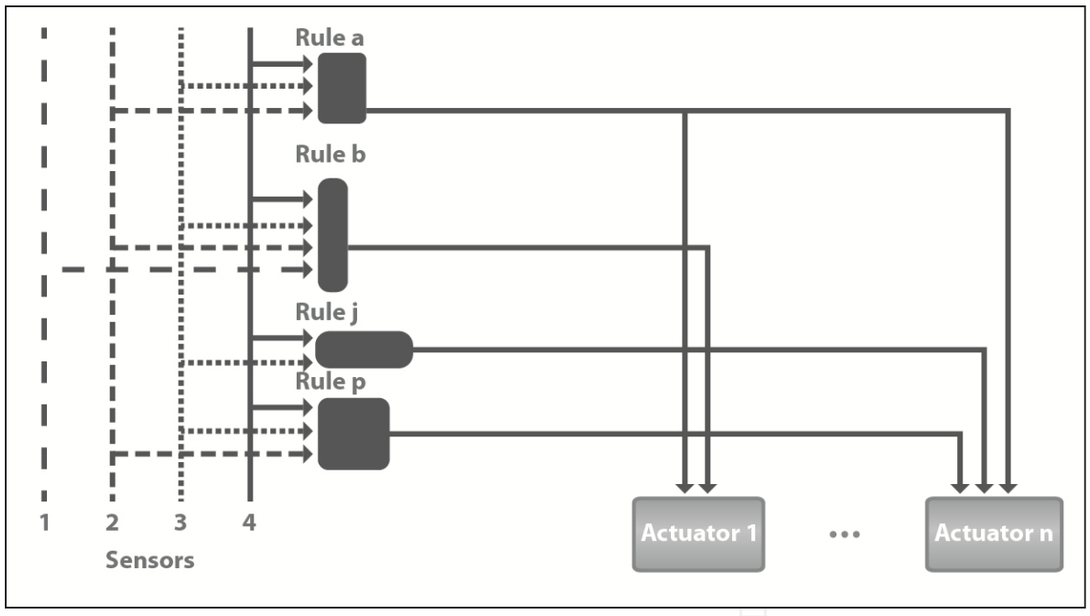


Fig. 6. Trigger mechanism of control rules.

The activation of a set of sensors could trigger different rules simultaneously, which leads to the necessity to define a strategy to trigger the rules. In the figure 6, the sensor S4 causes the activation of the rules a, b, j and p at the same time. Different rules may operate on the same actuator, which can generate an antagonising effect on the actuator, so it is necessary to recognize such situations.

The control rules management consists mainly of two phases, one to allow dynamic specification of the control rules and the other its activation:

Specification of control rules phase: The DCM allows remote users to edit a set of policies on devices if they have the security rights to do it. These devices have a longest life and can be reusable, because its control rules are persistent in a table of the database, allowing an adjustment in line with the interests and capabilities of the administrator. Control rules can be edited by the DSC manager, which allows distinguishing two instances to manage them:

1. As part of the assembly of components, default behaviour is defined and stored in persistent memory in the form of policies. Thereafter the administrator can define a set of control rules, which modify its behaviour. Rules should be evaluated and approved by the security administrator.

2. Whenever the DSC comes into operation, as a first step it checks for updates to the set of rules. In this case, new rules can be added or replace those predefined in the persistent memory and will be effective until a new set of rules is determined. The predefined rules are necessary to ensure a minimum performance in case the device cannot communicate with the business layer.

Activation of the control rules phase: when a device controller detects an event, the event is registered in the database. If there is a control rule associated to the event the commands are extracted from the database and the corresponding devices will receive instructions to perform the indicated actions.

As part of the activation mechanism of control rules, we define a server-type computer program called Events Controller whose fundamental role is to verify and apply existing control policies.

Figure 7 describes the protocol of activation of rules within DCM:

- The DSC communicates to the Events Controller the fulfilling of any condition.
- The Events Controller records in the database the received message.
- The Events Controller searches into the database to determine if there is associated a control rule, if so, obtains from the database the actions to be performed.
- The Events Controller could notify the user via a message system.
- The Events Controller orders the different actors to perform the corresponding actions that can be conceptualized as a chain of events.

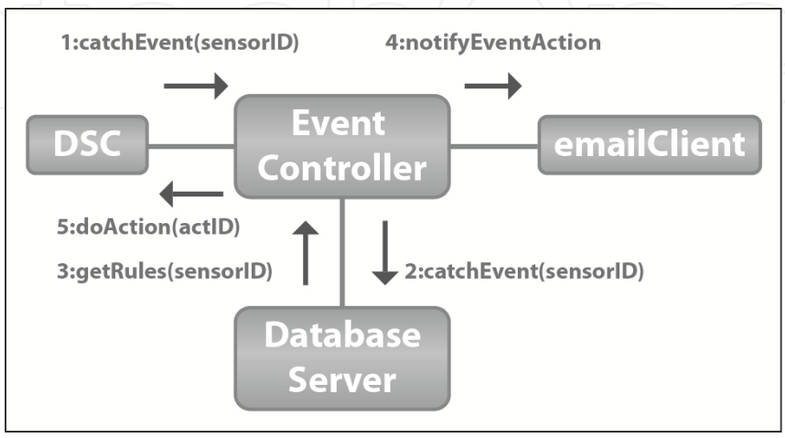


Fig. 7. Control rules implementation.

A device only performs the actions associated with an event when these are indicated by the Events Controller. However sometimes the device can perform an operation on its own initiative, a case is when the device loses communication with the controller.

There are two special cases identified in the rule triggering mechanism.

a. An event may trigger several rules at a time. There are two options to manage this situation:
   - To use the priority scheme within the rules as attributes, so if a set of rules can be selected, the choice of the rule will be according to the priority.
   - If a subset of eligible rules is still maintained, the mechanism can use a random activation sequence.
b. Many events interacting at the same time may run into contradictory actions, for example an event causes that an actuator is turned on and other event turns it off. A proposed solution to this case is to assign a priority scheme to events -according to a cost function-. These events are processed in accordance to the control rules management as follows:

Specification of control rules phase

- Identify rules for the same actuator
- Ask the user to classify them as contradictory or not
- If contradiction happens: the user must assign priority to events

Activation of the control rules phase

The actuator is empowered by the higher priority event when and if the event is in force.

4. Component-based design (CBD) used for embedded system design

To use the DCM in an embedded system, it is important to abide by the principles of component-based design, design contracts and resolve the problem of platform independence.

According to (Szyperski, 2002), a component must be understood as an executable unit, allowing the deployment and the composition at run time. This definition has been expanded from the one given in (Booch et al, 1999), where the component was conceived as the physical part of a system that meets and implements a series of interfaces. The definition also formalizes the idea given in (McIlroy, 1969). McIlroy first proposed the design of reusable component-based software, and also identified the need for catalogues of components, which are categorized according to different aspects.

Since McIllroy introduced this concept, efforts have been made to bring to the software industry, specific aspects of the culture of hardware production, but almost 40 years later there is still much to do. However, several evidences clearly show the convenience of using component-based design, both isolated in the software industry, as in embedded systems.

The component based design seeks to achieve the following benefits:

1. Reduce design and development time (using certified components).
2. Ensure the proper operation and performance of component-based software.

3. Simplify the evaluation of new components.
4. Increase the longevity of the investment associated with the design and construction of components.

Although the use of components is performed for a long time in the hardware industry, this situation does not occur in the software industry because there are conceptual differences about the use of components.

For example, they are product's catalogues which make possible the use of hardware components, and also international standards to assess the appropriateness of those components. In counterpart for using software components, catalogues are less available than those used in the hardware industry, and there are also various "universes search", where each universe corresponds to the programming language or platform used.

On the other hand while the control in the field of hardware can be sequential or parallel, and commonly there is no feedback about the success of the operation, in the field of software the control can be sequential or parallel, but almost always required to get feedback about the success of the operation.

In the field of embedded systems, they have a set of techniques that analyses the use of component-based design, for example Combest (Component-Based Embedded Systems Design Techniques) (Combest, 2007) states that to use the design based components within the field of embedded systems the following qualities are required:

- To allow heterogeneous components, combining the inherent capabilities that hardware has to work in parallel, with the usual capabilities of the software to work point to point, which should allow the existence of subsystems with execution and interaction semantics different.
- To have the ability to predict the basic properties of the designed system.

The authors of this chapter suggest that to achieve the correct composition of products in embedded systems is also required:

- Producers that are subject to rules to standardize parts.
- Clients that have access to similar parts catalogues.
- If required, the parts of a product can be transparently replaced by similar ones.
- Equivalent parts can be evaluated against other with similar properties and in function of the price.
- Time for the assembly of parts should be short.
- Predictability of the reliable performance of a component when assembled to others.

In the field of software engineering it is considered that the use of components promotes the simplification of processes, encapsulation, reusability, functionality, maintainability, efficiency, reliability and portability.

Under this strategy, all components communicate with its environment using two types of interfaces, an interface that describes the services provided (outgoing interface) and the other services required (incoming interface). However, the authors of this chapter consider that to design embedded systems based on components besides having a description of services, requires to have a description and parameterisation of the algorithms available and the data to be used.

4.1 Service description

The description of services is a common feature in software engineering issued in the software components and popularized by the use of Unified Modelling Language (UML) (Booch et al, 1999a), which although originally did not handle components, always described the amount and type of parameters and the type of return value (Booch et al, 1999b).

4.2 Algorithm description

Using software components allows to know both the services provided and required by a component, but it is not uncommon to find components that implement different algorithms, selectable by the user. As a black box a component hides commonly the implementation details of the methods. In this chapter it is stated that components should expose at least the name of the algorithms implemented and allow its parameterisation by the user.

4.3 Description of the data or signals that are sent and received

By merging the design of hardware and software it is important that the components used in embedded systems allow defining and configuring the physical layer, for example:

- meaning of the pins,
- domain and range of values in each pin,
- semantics of the value transferred
- transfer rate,
- representation of the data and
- ability to initialize variables

Other qualities of the components that the authors of this chapter consider desirable are:

1. to have a quick release of components on the market
2. to be able to certify the operation of the component
3. to have programming language independence
4. to establish, as part of deployment, the range of values on the signals to be handled, the encryption mechanism and the modulation of the signal to use.

The following features complement those mentioned by (Kopetz, 2007) as obligatory in embedded systems:

- to provide an efficient use of energy in devices that use batteries
- to have a low cost and time to market
- to have timely availability
- to have a dependable operation in safety-relevant scenarios

The authors of this document have identified some desirable qualities and other indispensable in embedded systems. Some of them are already present and others are missing, but despite this, the component-based design is now an option to the problem of designing embedded systems. Some identified solutions couple both disciplines, including:

1. Functional separation between inputs and outputs of the components.
2. Component models proposed by the software industry.
3. Standardized architectures for data migration between platforms.
4. Specialized programming languages, platform-independent, such as some based on IEC63131 standard (IEC, 2003), which despite its good intentions, has not yet reached its intended purpose.

One could also say that the evaluation of the implementation of new components via FPGA[1], to avoid the production of pieces of hardware in the initial phase of testing, allows prototyping the designs based on both technologies.

It should be noted that in designing embedded systems based on components, one must keep in mind the Principle of Compositionality, which states that the semantics of any component is determined by the semantics of the subcomponents, as well as the rules used to combine them. This is a fact to keep in mind when developing collaborative components and nested components.

4.4 Taking advantage of the component-based design into the DCM

The proposed model of distributed dynamic control (DCM) allows the separation of rules into two subsets as described in the previous section. This quality increases the longevity of embedded systems by allowing to partially modify their behaviour according to the user's requirements, to provide greater flexibility and adaptation.

The DCM allows dynamic creation of devices, taking advantage of the component-based design and the contract-based design. Rules within the DCM allow reaching cooperating devices, whose existence and location is dynamic, allowing modularity of devices. Nevertheless, the former does not ensure their compositionality. (Szabó, 2008; Zwiers, 1985) As a consequence there is a need to attend the embedded systems design with alternative arrangements to ensure the quality of the component, for example as in the contract-based design and the use of rigorous software testing.

The software industry is facing various problems that affect the development of embedded systems, including:

- To produce systems and component, which are highly efficient and robust.
- To ensure the quality of the applications and components developed.
- To eliminate side effects if software is reused.
- To facilitate the ability to leverage components and applications developed in a distributed manner, regardless of programming language and operating system for which they were developed.

To solve such problems various technologies of renowned quality have been proposed and have been using contract-based design (Meyer, 1992), software testing, component-based design (UML) and the interface definition language (OMG).

[1] A FPGA is a programmable logical device, of general purpose and configurable logic elements, making easy the prototyping of embedded systems, enabling to the user the design and testing of hardware components associated with software components already created.

Combining these technologies with FPGA allows achieving an environment for production of systems and devices of high quality that are easy to use and great capability to be reused.

4.5 Classes

To define an adequate functioning of a class and to increase its robustness, it is recommended to specify their behaviour and their status by using Contract-Based Design that defines pre-and post-conditions and invariant conditions defined in the classes. The control of these properties should be performed using software testing.

Although the individual quality of a class can be defined and ensured, in the context of interacting classes it is necessary to design and ensure the quality of these interactions. There are different practices and standards that divide business logic from the interfaces; this fact is reflected in several ways:

- Using the MVC pattern (Krasner et al, 1988).
- Using APIs to communicate developments made through time by various providers follows a set of specific standards, which would correspond to using the Bridge pattern (Gamma et al, 1995).
- Using Contract-Based Design (CrBD), which formalizes the provision of services between classes. All classes can define the pre and post conditions under which it can be provided a service, and also it can set some restrictions on its inner values.

4.6 Software components

Once the quality of a set of cooperating classes is assured, the set can be formalized to be reused later, giving rise to the software components.

The component-based design allows publishing the services that a component offers and also the services it requires, leaving all those hidden details of their operation, which in the best case facilitate the coupling of components and the replacement of any of them in a transparent way.

4.7 Interconnection between components

The need to combine software artefacts (whether packages, components or classes) that develop through a particular programming language for a specific operating system, led to the Object Managing Group to define the interface definition language (IDL) as a standard to regulate the definition of interfaces. Thus the IDL was created to solve the problem of the independence of programming language and operating system, allowing the definition of interfaces including embedded systems and wireless. Its implementation uses mapping between the IDL and the respective roles in different programming languages such as C, C + +, Java, Smalltalk, COBOL, Ada, Lisp, PL / 1, Python, Ruby and IDLscript.

Figure 8 shows the collaboration between components, where the weather station is responsible for providing information about weather conditions, which in turn are required by the device driver for a greenhouse. Once processed the signals and based on a set of rules, the controller may, for example, apply the irrigation of plants, opening windows to reduce temperature.

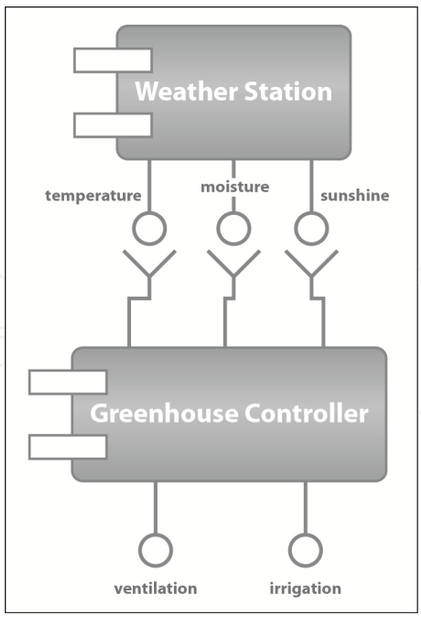


Fig. 8. Collaborating components.

From the implementation point of view, it could be a set of interacting classes used to monitor compliance with rules. For example list 1 shows one possible implementation of a unit that handles an invariant condition, and also shows the use of pre and post conditions.

```
public class Device{
  float lowValue_Warning;
  float topValue_Warning;
  float value;
  /** invariant value >=0 */

  public void setMaxValue(float v) throws UpperLimitError{
    /* require v>0; value<=topValue_Warning    */
    value+=v;
    if (value >= topValue_Warning)
      throw UpperLimitError;
  }

  public void setMinValue(float v) throws LowerLimitError{
 ...
      /* ensure v>0; value>=lowValue_Warning    */
  }
}
```

List 1. Using Contract-based design.

The code in list 1 might be associated with an IDL definition as shown in the list 2.

```
module DeviceApp {
   interface Device
   {
      // Under level exception
      exception LowerLimitError {};

      // Over level exception
      exception UpperLimitError {};

      // Record a new upper level
       void setMaxValue(in float v) raises (UpperLimitError);
       void setMinValue(in float v) raises (LowerLimitError);;
   };
};
```

List 2. IDL definition.

5. Case studies: MCD support in greenhouse management

The control rules shown below illustrate the relationship between configuring and monitoring various sensors and actuators in the production of flowers and foliage. To exemplify the control rules in the cultivation of flowers we take common concepts in the orchid's production of Phalaenopsis type. To propose control rules in the foliage production, we use examples about the coriander (Eryngium foetidum).

The example consists of three phases:

1. Definition and parameterisation of variables
2. Simple rules supported by MCD
3. Associated rules with production process supported by MCD.

5.1 Phase 1: Definition and parameterization of variables

The variables definition and initial values for temperature control, humidity and the period of lifetime of a process are described below.

5.2 Humidity

Since the low or high humidity levels impact the plant growth, the humidity control is very important. High humidity inhibits leaf transpiration, which restricts the movement of nutrients in the plant, producing a fungal infection and also the death by malnutrition. On the other hand, low humidity reduces water absorption by roots. In the examples, the values assigned to humidity are between acceptable ranges. For example

```
hum_max = value2;
hum_min = value1;
```

5.3 Temperature

Temperature is a relevant variable to which plants react. The temperature is controlled by a range defined by a maximum and a minimum value, and the interval in Celsius degrees required to warn of extreme conditions.

E.g.

```
temp_max    = 28; //Maximum limit acceptable for temperature
temp_min    = 20; //Minimum limit acceptable for temperature
delta_Temp  = 3; //range to alert the proximity to the temperature limit
```

5.4 Time

The timer has great relevance because planting, care and production is given in accordance with specific deadlines for each crop. E.g.

```
startDate   = setDate("yy1/mm1/dd1");
endDate     = setDate("yy2/mm2/dd2");
today       = getDate().getDay();
hour        = getDate().getHour();
day         = (hour > 6) && (hour<17:30)?true:false;
night       = (day)?false:true;
nsec        = # of second that activate the actuator
```

5.5 Phase 2: Simple rules supported by MCD

To describe simple rules supported by the MCD, we use the experiences of diverse researches, who have studied the Eryngium foetidum crop (Alvarado-Sanabria & Villalobos, 1999; Morales, 1995; Santiago-Santos & Cedeño-Maldonado, 1991), and we have adapted some of their experiences in the form of rules, that we describe in Table 1.

Notice in these examples that the rules are described by the combination of 4 attributes that handle: type of sensor that is controlled (in this study we have used wireless sensors connected via ZigBee protocol), the control rule (condition of interest with respect to the sensor at any given time), actions to be taken when the value reported by the sensor allow the condition to be controlled, and a message that can be mailed to an e-mail address, or mobile device.

Sensor	Control rule	Action	Message
temp	temp<=15ºC	lighting system(on)	
soil humidity	hum == 50%"	message system(send)	"hum_min=50%"
humidity	h<60%	irrigation_drip system(on, nseg)	
light	luminosity >80%	curtain system(close)	
timer	day=HARVEST_DAY	message system(send)	" time to harvest"

Table 1. Examples of simple rules to control the Eryngium foetidum crop.

5.6 Phase 3: Associated rules with production process supported by MCD.

Phase 2 described the support that the MDC gives to management the simple control rules; this section describes how the MCD also allows rules to obey production strategies, where the iterative processes specified in the pseudo-code are defined in the MCD with respect to time.

The following examples show how in the MCD is defined a production strategy according to expected demand. Here are some basic processes of interest to the owner of the greenhouse.

Controlling temperature fluctuations between day and night

The Phalaenopsis orchids can be forced to flourish when the temperature is reduced by more than 5 degrees at night for a period of 2 to 4 consecutive weeks, but if the flowering of the orchids needs to be inhibited, it is necessary to maintain a constant temperature of 29 Celsius degrees during the same period of time (Blanchard & Runkle, 2006). In the example above it is assumed that to control the temperature the glasshouse required to have windows, ceiling fans, lighting system and messages system.

```
if (temp >= (temp_max(day) - delta_Temp)) {  // Temperature approaches its maximun
    fan(on)
    window(open)
}
if (temp <= (temp_min(night) - delta_Temp)) { //Temperature approaches its minimun
    send_message("abnormal temperature reduction ")
    window(close)
}
if (temp == (temp_min(night))) {                    //Temperature temperature reaches its
                                                    // minimum level
    light (on)
}
```

Check the average temperature over a period of several days

Control the harmony in temperature can lead to better results in the production of some flowers; hence the following pseudo-code would monitor this condition:

```
while ((startDate<today) && (endDate>today)){
    controlTempMax(temp_max);
    controlTempMin(temp_min);
}
```

Subjecting a crop to water stress or temperature stress produces alterations in the production cycle of flowers. In that case, the MCD contributes to maintain the desired conditions. E.g.:

```
for (int i=startDate;i <=endDate;i++){
    temp_maxeratura [i] = temp_max;
    temp_mineratura [i] = temp_min;
}
```

```
while ((startDate<today) && (endDate>today)){
    controlTempMax(temp_maxeratura[today]);
    controlTempMin(temp_mineratura[today]);
}
```

Other methods that could be implemented include:

- A temperature plan to establish the desired colour of the flowers.
- A temperature plan to produce more flowers a particular day of the week.

6. Acknowledgment

This work has been supported by Spanish Government, in particular by "Agencia Española de Cooperación Internacional para el Desarrollo"under funds from D/027406/09 for 2010, D/033858/10 for 2011, and A1/039089/11 for 2012 and by the Laboratory in Automate and Intelligent Systems in Biodiversity of the University of Costa Rica.

7. References

Alvarado-Sanabria, C. & Villalobos, J. (1999) El culantro de coyote (Eryngium foetidum cursiva) para exportación. Ministerio de Agricultura y Ganadería, Costa Rica. 1999

Blanchard, Mathew & Runkle, Erik. (2006) Temperature during the day, but not during the night, controls flowering of Phalaenopsis orchids. Journal of Experimental Botany. Vol 57. Issue 15, pp4043-4049.

Booch, G., Rumbaugh J. & Jacobson, I. (1999a) The Unified Modeling Language User Guide. Addison-Wesley, Reading, MA

Booch, G., Rumbaugh J. & Jacobson, I. (1999b) The Unified Modeling Language for Object-Oriented Development. Documentation Set Version 0.9a Addendum. Retrieved from: http://www.ccs.neu.edu/research/demeter/course/f96/readings/uml9.pdf

Combest. (2007) Ist Strep COMponent-Based Embedded Systems design Techniques. Scientific Foundations for Component-based Design, and their Limitations Component-based Design for Embedded Systems. Seventh Framework Programme for Research and Technological Development (2007-2013). Retrieved from: http://www.combest.eu/home/?link=CBDforESMcIlroy, Malcolm Douglas (1969). "Mass produced software components". Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968. Scientific Affairs Division, NATO. p. 79.

Gamma, Erich; Helm, Richard; Johnson, Ralph & Vlissides, John. (1995) Design Patterns: Elements of Reusable Object-Oriented *Software*. Pearson Ed. Madrid. 1995

IEC International Electrotechnical Commission. (2003) IEC 61131. Programmable controllers. Retrieved from: *http://webstore.iec.ch/preview/info_iec61131-1{ed2.0}en.pdf*

Krasner, Glenn & Pope, Stephen. (1988) A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object Oriented Programming, 1(3):26-49, August –September 1988.

Kopetz, Hermann. (2007) The Complexity Challenge in Embedded System Design. Research Report 55/2007. Retrieved from:

http://www.vmars.tuwien.ac.at/courses/dse/papers/complexity.pdf

Meyer, Bertrand. Applying Design By Contract. (1992). IEEE Computer, Vol. 25, No. 10, pp 40-51.

Morales, José Pablo. Cultivo del cilantro, cilantro ancho y perejil. (1995) Fundación de desarrollo agropecuario, Inc. Serie Cultivos. Boletín Técnico #25. Santo Domingo.

Santiago-Santos, Luis R. & Cedeño-Maldonado, Arturo. (1991) Efecto de la intensidad de la luz sobre la floración y crecimiento del culantro, Eryngium foetidum. Journal of Agricultura of the University of Puerto Rico. Oct 1991, v 75(4) p. 383-389

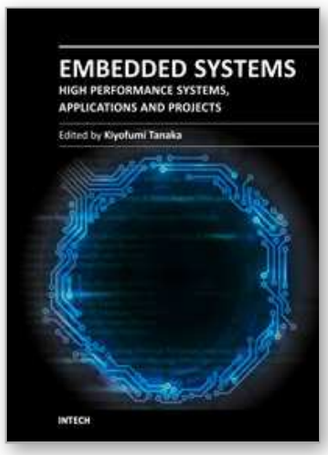
Schneeman, R.D. (1999). Implementing a Standards-based Distributed Measurement and Control Application on the Internet. In : U.S. Department of Commerce.

Szabó, Zoltán Gendler, "Compositionality" (2008), *The Stanford Encyclopedia of Philosophy (Winter 2008 Edition)*, Edward N. Zalta (ed.). Retrieved from: http://plato.stanford.edu/archives/win2008/entries/compositionality/.

Szyperski. Clement. Gruntz, Dominik & Murer, Stephan. (2002) Component Software - Beyond Object-Oriented Programming. Addison-Wesley

Vásquez Morera, Javier; Vásquez, Núñez, José Luis & Travieso González, Carlos M. (2010) Changing control rules dynamically in embedded digital control systems. WSEAS Transactions on Systems and Control. Issue 3, Volume 5, March 2010

Zwiers, Job; de Roever, Wilhem Paul & vam Ende Boas, Peter. (1985) Compositionality and concurrent networks: soundness and completeness of a proof system in Proc. of 12th colloquium on Automata, Languages and Programming, W. Brauer (ed.), LNCS 194, Springer-Verlag, 1985.



Embedded Systems - High Performance Systems, Applications and Projects

Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0350-9
Hard cover, 278 pages
Publisher InTech
Published online 16, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permeated various scenes of industry. Therefore, we can hardly discuss our life or society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 13 excellent chapters and addresses a wide spectrum of research topics of embedded systems, including parallel computing, communication architecture, application-specific systems, and embedded systems projects. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book as well as in the complementary book "Embedded Systems - Theory and Design Methodology", will be helpful to researchers and engineers around the world.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen