SCA Extensions to Support Safety Critical Distributed Embedded Systems

Aitor Agirre, Jon Perez
IK4-Ikerlan
Arizmendiarieta 2
20500 Arrasate, Spain
aagirre, jmperez@ikerlan.es

Rafael Priego, Marga Marcos
Dept. of Automatic Control and
Systems Engineering
ETSI Bilbao, UPV/EHU
Bilbao, Spain
rpriego001, marga.marcos@ehu.es

Elisabet Estévez
Dept. Of Electronics and
Automatic Control
EPS Jaén, Universidad de Jaén
Jaén, Spain
eestevez@ujaen.es

Abstract

Component Based Software Engineering (CBSE) is being increasingly applied in the distributed embedded systems (DES) domain as long as these systems are getting more and more complex in terms of flexibility, dynamism or heterogeneity. Besides that, safety critical systems must cope with the fulfillment of safety requirements and certification standards. This factor increases considerably the development cost of safety distributed embedded systems, even more if they must cope with flexibility, dynamism and heterogeneity.

This paper focuses on the distribution aspects of such systems, and more specifically on safe communication channels for safety critical distributed systems. The proposed approach describes a certifiable general purpose safety communication layer that could be reused in different systems, thereby reducing the cost of system development and certification.

1. Introduction

Safety critical embedded systems are dependable systems that could cause significant economical loss, injury or loss of human life in case of failure. The international safety standard IEC-61508 [1] is a generic safety standard for industry that has been used to define multiple domains specific standards such as industrial process, railway, automotive, etc. The development and certification of distributed safety critical embedded systems is expensive [2, 3] and dynamic behavior or functionality must be carefully analyzed and justified.

One key aspect in distributed embedded systems is the communications related software [4]. Sometimes it is implemented from scratch using low level mechanisms like raw sockets or even OSI layer level 2 protocols like EtherCAT or CAN. Some other times, standardized distribution middleware technologies are used, either generic (Java RMI, SOAP Web Services, CORBA…) or adapted to the embedded domain (CORBA/e [5], REST, ICE-e [6] or DDS [7]). In this sense, considerable research effort has been invested in providing QoS support to this distribution related software layer, frequently focused on real time and resource management constraints [8, 9] rather than dependability or safety aspects.

The availability of a certifiable, generic and reusable distribution middleware with appropriate safety functions and properties might ease and reduce the development cost of distributed safety critical systems. With these aspects in mind, it seems reasonable the application of CBSE principles [10] to this kind of distributed systems with safety requirements.

Alongside this main objective, several other requirements have been considered in the design of the proposed safety communication layer: Ease of use in terms of abstraction from the low level communication details, flexibility, modularity, platform independence and of course those related to the fulfillment of the safety mechanisms described in IEC 61784-3-3 [11].

The IEC 61784-3-3 standard (section 3.1.1.2) defines “black channel” as a communication channel without available evidence of design or validation according to IEC 61508 series (e.g. Ethernet). In other words, a black channel is an unsafe communication channel that requires a safety communication layer to ensure safe communication among distributed components. To achieve this objective, the standard defines a set of techniques and measures to detect all possible errors (data corruption, incorrect order, message lost, message arriving outside temporal requirements, etc.) that could be caused by different reasons (faults) such as:

- Random malfunctions (e.g. EMI impact on the transmission channel).
- Failures/faults of the standard hardware.
- Systematic malfunctions of components within the standard hardware and software.

To abstract these safe communication techniques and at the same time fulfill the previously exposed non-functional requirements of flexibility, modularity and platform independence, the SCA (Service Component Architecture) component model has been adopted for the business components implementation, due to several features it provides [12].
The rest of the paper is organized as follows: Section 2 presents an overview of the techniques proposed in the IEC 61784-3-3 [11] standard to achieve safe communication over a black channel. In Section 3 a brief description of the key features and concepts of SCA is provided. Section 4 presents an approach, based on SCA extensions, to fulfill the IEC 61784-3-3 safety requirements in a communication system built on top of a black channel.

2. Black channel protection mechanisms

The IEC 61684-3-3 describes several techniques and measures to perform a safe communication over an unsafe transmission media, i.e. a black channel. In this preliminary work, and as a proof of concept, only three diagnosis techniques have been considered in order to check the feasibility of the proposed approach. They are included in the Table 1 below, alongside with the errors they detect.

<table>
<thead>
<tr>
<th>Error</th>
<th>Diagnosis technique</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq. number Watch-dog</td>
</tr>
<tr>
<td>Corruption</td>
<td></td>
</tr>
<tr>
<td>Repetition</td>
<td></td>
</tr>
<tr>
<td>Incorrect seq</td>
<td></td>
</tr>
<tr>
<td>Loss</td>
<td></td>
</tr>
<tr>
<td>Delay</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Safety related techniques.

The safety measures provided by the safety communication layer have to ensure that the data from the sender arrives to the receiver reliably, without repetition or delay, in an ordered way, and with no change or modification in the transmission. This is achieved through the following measures:

1. **Watch-dog**: the data sampling rate often has to be guaranteed in critical control applications. In this sense, a deadline can be defined for each message type through a watch dog, which detects timeouts in the reception of messages.

2. **CRC signature**: Prevents message corruption in the transmission media, which may be due to multiple causes, ranging from noisy wires to defective hardware. For every message written to the wire, the CRC of the message payload is computed and attached to the message itself.

3. **Sequence number**: To detect message loss or repetition (intended or unintended), the sender attaches a sequence number to the message to be transmitted. This sequence number is increased with each transmission, and this way, the receiver can verify that the messages arrive in order, with no repetition or packet loss in between.

3. SCA component model

   It seems obvious that one of the key points in the CBSE is the selection of a proper component model for our needs. In previous work [12] SCA was selected due to several reasons: Besides providing modularity, flexibility and reusability as many other component models [10], in SCA the decoupling of functional and non functional aspects is perfectly clear, the application deployment can be standardized in a relatively easy way and it supports declarative reconfiguration of the applications. Additionally, it provides extension points that make the runtime extensible and highly customizable, which is a key feature for our needs.

   SCA architecture is defined in a set of standards that define a programming model for building applications from building blocks (components) that encapsulate the business functions and offer them as services. That is, SCA provides solutions based on a Service Oriented Architecture paradigm.

   A graphical representation of an SCA component is depicted in Figure 1. An SCA component comprises an implementation and a set of properties, services and references. The component implementation is language agnostic, and depending on the SCA distribution, more or less languages can be available. The SCA standard includes C, C++, BPEL, Java and Spring, but distributions as FraSCAti [13] or the de facto reference SCA implementation Apache TUSCANY [14] add additional implementation types such as Java scripting languages, Scala or OSGi.

   ![Figure 1. SCA component diagram.](image)

   The services and references can be specified through an extensible set of interface types (Figure 2): Java interfaces, WSDL (Web Services Description Language), UPnP (Universal Plug and Play), C headers and so on. Additionally to the interface type, a specific binding can be specified for a service or a reference. The binding concept refers to the distribution middleware that will be used to interconnect a service with a reference. If nothing is specified, the default SCA binding will be used, which can be implemented in different ways by different SCA runtimes [15].
Currently the standard includes the Web Services, JMS (Java Messaging Service) and JCA (Java EE Connection Architecture) bindings, but new bindings can be easily added to the runtime. In this sense, distributions as TUSCANY or FraSCAti provide bindings for CORBA, REST RMI or ATOM.

Alongside with the binding concept, SCA provides support for specification of constraints, capabilities and QoS expectations from component design to concrete deployment. Following the AOP (Aspect Oriented Programming) paradigm, the SCA policy framework allows intents and policies to be associated with SCA components [16]. This way, the components can declare the non-functional services they depend upon, and let the runtime guarantee these policies are enforced [17]. So far, the SCA policy standard includes security and transactions, but the set of supported policies can be extended to meet user specific needs (e.g. persistence, logging, safety).

All the previous concepts that compose an SCA component, together with the component interconnections ("wires"), are declaratively specified in an XML file with .composite extension. Figure 2 represents a composite that is composed by two components wired together.

To summarize, SCA provides several extension points that make the runtime highly configurable and extensible: (1) implementation types, (2) interface types, (3) bindings and (4) policies. The proposed approach for safety critical distributed systems is based in these extension capabilities, and more specifically in two of them: A DDS binding and a safety policy.

4. Proposed SCA extensions

The upper drawing in Figure 3 represents a simplified ad-hoc approach for a safety critical distributed application that ensures safe data communication over a black channel without a generic safety communication layer. Components in gray are safety related.

In such simplified ad-hoc approach, the responsibility of implementing the safety related mechanisms stated in IEC 61784-3-3 standard (see Table 1) relies in the application business components. This fact usually leads to ad-hoc solutions where typically a small part of the communications related software is reusable and most of the times its API is communications oriented, thus providing a low level abstraction of the communications related stuff.

In contrast, the proposed approach is based in a generic layer that provides all the safety related mechanisms detailed in Section 2 in a transparent way for the developer. This safety communications layer is based in two SCA extensions:

(1) DDS binding: DDS is a distribution middleware which is highly configurable in terms of QoS parameters. Aspects like communication reliability, network bandwidth usage or message latency can be easily configured declaratively through a configuration XML [18]. It natively supports the watch-dog mechanism described in Section 2 through the deadline QoS parameter. Additionally, it can be implemented over any OSI level 2 transport layer, e.g. CAN [19], which makes it a suitable option for distributed embedded systems.

(2) Safety policy: The SCA policy can be declaratively attached to the DDS binding. It includes the CRC signature and sequence number safety mechanisms detailed in Section 2. For each business operation invoked from a component reference, the safety policy computes the CRC signature of the original DDS topic, which includes the operation name plus its arguments and a sequence number. Then, the CRC signature is attached to the original DDS topic as an additional field, and that “signed” topic is finally written to the wire. In the service side, the signed topic is received and the process is reversed: The receiver computes the CRC signature of the original information and compares it to the received CRC, thus ensuring the integrity of the received message. On the other hand the sequence number prevents message loss and repetition (intended or unintended).

This approach completely abstracts the developer from the communications related issues. There is no more need for communications oriented API-s, as far
as the binding layer wraps the business interfaces and ties them to the appropriate generic communication code underneath. The developer can focus on the business logic and interfaces as if it were a non distributed application. The only thing to specify in the SCA .composite file is the binding and policy to be used in the reference and service sides of the "wire", e.g. for the reference side:

```xml
<reference name="myRef">
  <binding.dds requires="safety" topicName="Data"/>
</reference>
```

Note that the binding could be also used with no safety policy in non safety critical applications, thus saving the computation cost of safety related calculations, e.g. CRC.

The requires keyword indicates to the SCA runtime that the safety policy must be injected in the tagged binding, thus attaching to that binding interface the safety mechanisms implemented in the policy.

5. Conclusions and future work

A generic safety communication layer for safety critical DES has been proposed based on two SCA extension mechanisms, namely a DDS binding combined with a policy that implements safety related mechanisms described in IEC 61784-3-3 standard.

This approach pursues two main objectives: (1) Safety critical systems development cost reduction through reusability of a certifiable safety communication layer, and (2) enforce developer efficiency through the abstraction of distribution related code.

Future work aims to extend the DDS binding to support client-server interaction paradigm (currently only publish-subscribe pattern is supported) and include more safety mechanisms in the safety policy.

References


