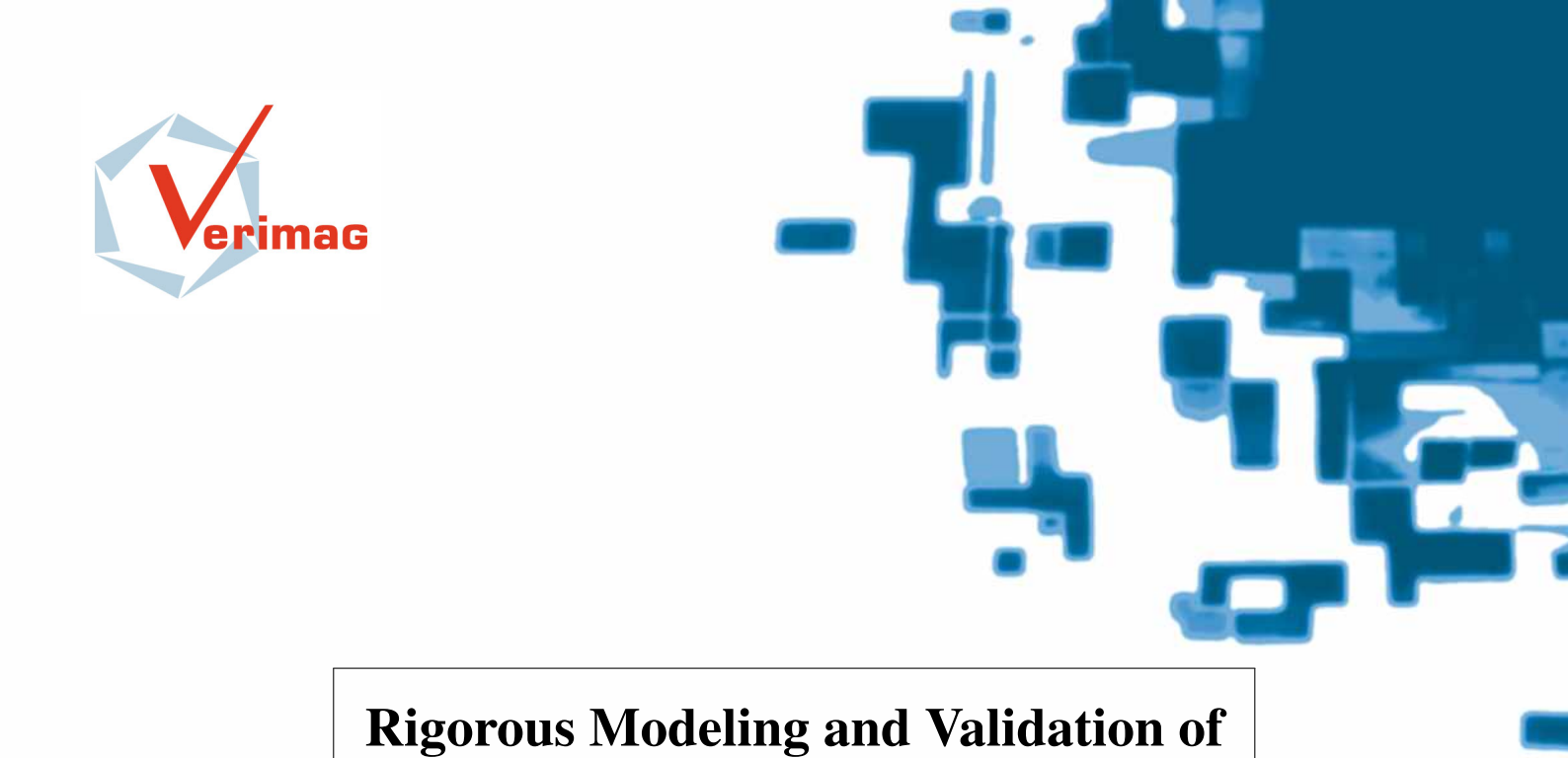


Rigorous Modeling and Validation of CANopen Systems

Alexios Lekidis, Marius Bozga, Saddek Bensalem

Verimag Research Report n^o TR-2014-1

January 23, 2014

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Rigorous Modeling and Validation of CANopen Systems

Alexios Lekidis, Marius Bozga, Saddek Bensalem

January 23, 2014

Abstract

CANopen is an increasingly popular protocol for the design of networked embedded systems. Nonetheless, the large variety of communication and network management functionalities supported in it can increase significantly systems complexity and in turn, the needs for system validation at design time. We present hereafter a novel approach relying on modeling and verification techniques, allowing to provide a comprehensive and faithful analysis of CANopen systems. This approach uses BIP, a formal framework for modeling, analysis and implementation of real-time, heterogeneous, component-based systems and the associated BIP tools for simulation, performance evaluation and statistical model-checking. As a proof of concept, the approach has been applied in an existing benchmark simulating a realistic automotive system and the Pixel Detector Control System of the ATLAS experiment at CERN's Large Hadron Collider (LHC) particle accelerator. This work facilitates the systematical development of such systems and reveals potential application for the generation of optimal device configurations.

Keywords: Networked embedded systems, Fieldbus protocols, Model-based development, Performance evaluation, Formal validation

Reviewers: Marius Bozga

How to cite this report:

```
@techreport {TR-2014-1,
  title = {Rigorous Modeling and Validation of CANopen Systems},
  author = {Alexios Lekidis, Marius Bozga, Saddek Bensalem},
  institution = {{Verimag} Research Report},
  number = {TR-2014-1},
  year = {}
}
```

Contents

1 Introduction

Fieldbus protocols provide efficient solutions to important issues occurring in embedded system design nowadays. Such issues include managing system complexity, reducing communication cost as well as providing guarantees for functional and real-time requirements. A high-level protocol included in this family is CANopen [13]. This protocol is getting increasing popularity thanks to a vast variety of communication mechanisms, such as time or event-driven, synchronous or asynchronous as well as to additional support for time synchronization and network management. Moreover, it provides a high-degree of configuration flexibility, requires limited resources and has therefore been deployed on many existing embedded devices.

Applications using CANopen as their communication protocol can be found in automotive systems. In this domain, it is used as a high-level protocol on top of Controller Area Network (CAN) [5], in order to organize and abstract its low-level communication complexity. Since it offers parametrization according to predefined standards and manufacturer-specific device specifications, CANopen has also been deployed in distributed control systems, maritime electronics, medical devices, railway applications, photovoltaic and building automation systems e.t.c.

To the best of our knowledge, the existing tools for simulation, analysis and validation of CANopen systems are very limited. Vector GmbH [1] provides a powerful tool for the simulation of such systems, the CANalyzer [27]. It also contains a CANopen extension, namely the CANalyzer.CANopen. A relevant

[1] https://vector.com/

tool can be found in youCAN stack prototypes [23], provided by port GmbH [2]. CANopen Magic [16] is an interactive tool from Embedded Systems Academy [3], providing an interface for the development and simulation of applications using the protocol. Nonetheless, these tools are not able to perform timing analysis and validation. Furthermore, their use in the design of correct, functional CANopen systems requires high expertise. Likewise, since they are targeting an industrial use, their evaluation versions can only be used to familiarize with the protocol. Subsequently, they have limitations on the network size and the protocol functionalities. On the other hand, the equally powerful tools for CAN, capable of performing both timing analysis and performance evaluation, such as RTaW-Sim [21], are not implementing the CANopen protocol.

The aforementioned considerations are present, because CANopen is a fairly complex protocol and the interactions between the different types of communication mechanisms are very subtle. Thus, the protocol primitives can be easily misused, leading to poor, non-functional systems. A particular example is that even though it offers a wide range of services and communication mechanisms, the proper use of them is left to the device manufacturer as well as the application developer. The default method of setting the protocol parameters does not apply in many cases. An efficient solution to this issue is the availability of validation support at design time. Previous studies [1] [24] have illustrated that conformance testing can be used as a validation method in CANopen systems, due to its capability of verifying system integrity as well as the protocol's error-free functionality. However, the lack of functional error detection and performance analysis in earlier stages of the development cycle is a considerable design limitation for such systems.

In this work we present an alternative validation method based on the use of formal modeling and verification techniques. We provide a systematic way to construct models of CANopen systems using the BIP component framework [2]. These models are constructed systematically, using a structural translation principle, from the protocol's entities and communication mechanisms. We show that the models obtained are faithfully compliant with the protocol's functional and timing aspects and can be used for either functional verification or performance analysis, using existing simulation and statistical model-checking tools available for BIP. As far knowledge is concerned, this method is the first attempt to obtain formal models for the CANopen protocol.

The rest of this article is organized as follows. Section 2 presents a brief introduction of CAN and CANopen. Section 3 introduces briefly the BIP framework along with its associated tools and discusses the techniques used for verification. Section 4 introduces the modeling and structural translation principles of CANopen systems in BIP. Section 5 provides experimental results of the model-based translation on existing benchmark systems and discusses current validation issues. Finally, Section 6 provides conclusions and perspectives for future work.

2 Overview of CAN and CANopen

2.1 Controller Area Network (CAN)

The Controller Area Network (CAN) was initially introduced in the beginning of the 1990s by Robert Bosch GmbH targeting industrial automotive control systems. Its scope was to reduce the number of wires in passenger cars through a serial Bus system. Nonetheless, its use gradually became wider and nowadays it is also found in various distributed embedded systems.

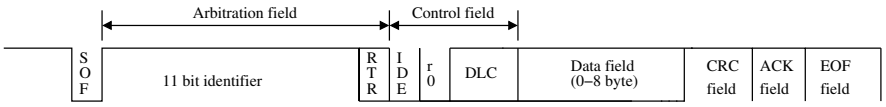
CAN is a message-oriented transmission protocol, based in a multi-master access scheme to a shared medium. It uses the Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) approach, in order to solve bus contentions deterministically. Its protocol stack implements only the physical and the data link layer of the OSI reference model, thus reducing the message processing delays and simplifying the communication software. The physical layer is responsible for data transmission, whereas the data link layer for managing the access on the Bus. CAN assigns a unique identifier to each transmitted message, which defines both the content and the priority of the message. The absence of originating or destination addresses facilitates the addition of new nodes in the network, without stopping its operation. Another advantage is that the network allows multi-casting capabilities.

[2] http://www.port.de/
[3] http://www.esacademy.com/

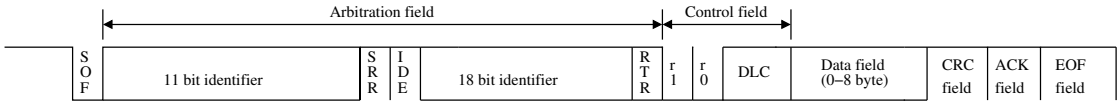
CAN messages are denoted as frames. Each frames can be transmitted only when the Bus is idle. They are of three types, namely:

- Data frame, used for data transmission
- Remote frame, used for data request
- Error frame, used to report error conditions

Data frames are also divided in the standard and the extended format. Their main difference is found in the frame identifier. The former (Figure 1a) defines a 11-bit identifier supporting up to 2048 different frames in the network, whereas the latter (Figure 1b) a 29-bit identifier supporting up to 536870912. Remote frames are similar to data frames, though they do not carry data, and error frames consist of an error flag, denoting the occurred error condition and an error delimiter.



(a) Standard CAN frame



(b) Extended CAN frame

Figure 1: Classic CAN data frame format

The transmission of each frame field is followed by a synchronization between all the connected nodes in the network and the Bus, during which the latter broadcasts the received data to all of them.

The beginning of every frame, except the error frames, is indicated by the Start Of Frame (SOF) field, which corresponds to 1 bit. Immediately after the SOF, is the Arbitration field containing the frame identifier identifier and the Remote Transmission Request (RTR) bit. The frame with the lowest identifier is always transmitted first, since the dominant level (binary 0) in CAN has higher priority than the recessive (binary 1). A dominant value in the RTR bit denotes a data frame and a recessive a remote frame, ensuring higher priority for the former. The Control field contains the Identifier Extension (IDE) bit, the reserved r0 bit as well as the Data Length Code (DLC) field. The IDE bit distinguishes a standard from an extended data frame. In the extended frame format the Arbitration field is larger, due to the addition of the IDE bit and the Substitute Remote Request (SRR) bit. Moreover, the Control field includes one more reserved bit in the place of the IDE bit. The DLC field denotes the length of the data and its value is between 0 and 8. The Data field, not applicable in the remote frame, contains the frame data, which are according to [5] from 0 to 8 bytes. The integrity of a frame is guaranteed by the Cyclic Redundancy Check (CRC) field, consisting of 15 bits plus a 1-bit delimiter. The ACK Field, consisting of 2 bits (ACK bit and a ACK delimiter), serves as an acknowledgment, if at least one node received the frame successfully. In the opposite case, the frame will be retransmitted consecutively until it is successfully received. Finally, the End Of Frame (EOF) field indicates an error-free transmission to all the nodes connected in the network and corresponds to 7 recessive bits.

CAN provides a high-degree of synchronization among the connected nodes, due to the sufficient number of edges in the transmitted bit sequence. This is achieved by a bit encoding technique, termed as bit-stuffing [5]. According to it, whenever 5 consecutive bits have the same value (dominant or recessive) an additional bit of the opposite value is added to the sequence. However, the added bit can be followed by a sequence of 4 consecutive bits of the same polarity. Thus, in the worst case the number of consecutive bits subject to bit-stuffing is considered as 4. Though bit-stuffing is an important and efficient technique, it increases the frame response time proportionally to the content of the transmitted data and therefore is not fixed. A highly probable outcome of its usage are the additional transmission jitters, which may cause deadline violations.

As each node receives every transmitted frame on its local receive buffer, a frame acceptance filtering has to be applied. This mechanism determines if the received data are relevant to the specific node or not. In the first case, the frame is sent to the upper communication layer, whereas in the second it is discarded.

The ability to resolve collisions deterministically is one of the protocol's main characteristics. This is accomplished through the arbitration mechanism, following the transmission of the SOF bit. The outcome of this technique is to ensure that only the node with the highest priority frame will transmit its data to the Bus. This process is serial, meaning that the frame's identifier will be transmitted bit-per-bit. The level of the Bus will be dominant if at least one node is transmitting a dominant bit. If a node is transmitting a recessive value and senses the Bus at dominant level, it will immediately halt, since it will understand that it lost the contention. It will only retry whenever the current frame transmission ends and accordingly senses the Bus idle again. However, if two or more nodes having the same frame identifier proceed to the arbitration phase at the same time, unmanageable collisions will occur in the Bus. The only exception for this situation is found in remote frame transmissions, where the involved data request frames will just overlap. Nevertheless, the requesting node is usually unaware of the data length he is about to receive and sets the DLC field randomly. Consequently, unmanageable collisions are still not avoided.

Although CAN is robust and cost-effective, its low-level complexity and the correct allocation of the frame identifiers introduce certain obstacles in complex CAN-based system design. To facilitate their design, high-level protocols build on top of CAN, such as CANopen [13] for embedded systems, DeviceNet [25] for factory automation and J1939 [26] for trucks and other vehicles have been proposed. All these protocols except CANopen are found only in CAN-based systems. Whenever used in such systems, they adopt the CAN standard frames, since they are shorter and enable higher communication efficiency.

2.2 CANopen

CANopen uses a master/slave architecture for management services, but concurrently allows the utilization of the client/server communication model for configuration services as well as the the producer/consumer model for real-time communication services. A comprehensive introduction to the protocol can be found in [22]. Unlike other Fieldbus protocols it does not require a single master controlling all the network communication. Instead a CANopen system is specified by a set of devices (Figure 2), which in turn use a set of profiles, in order to define the device-specific functionality along with all the supported communication mechanisms. The communication profile defines all the services that can be used for communication and the device profile how the device-specific functionality is made accessible. The communication profile is defined in the DS-301 standard [13], whereas the device profiles providing a detailed description on CANopen's usage for a particular application-domain, are defined in the DS-4xx standards [4]. If CANopen systems require configurations or data access mechanisms not covered by the standard communication profile, profile extensions can also be defined. These are called Frameworks and are found in the DS-3xx standards [4].

The protocol's communication mechanisms according to the DS-301 are specified by standard *Communication Objects (COB)*. All the COBs have their own priority and are transmitted through regular frames of the chosen lower-layer protocol. They are generally divided in the following main categories:

- *Network Management objects (NMT)*, used for the initialization, configuration and supervision of the network
- *Process Data Object (PDO)*, used for real-time critical data exchange
- *Service Data Object (SDO)*, used for service/configuration data exchange
- *Predefined objects*, found in specific entries in every OD. The featured objects in this category are:
  - *Synchronization object (SYNC)*, broadcasted periodically to offer synchronized communication as well as coordinate operations
  - *Timestamp object (TIME)*, broadcasted asynchronously to provide accurate clock synchronization using a common time reference

[4] http://www.can-cia.org/index.php?id=440

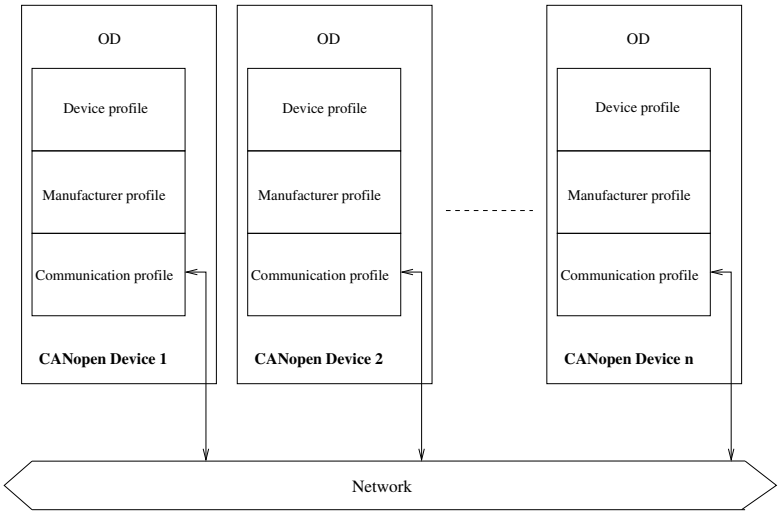


Figure 2: Communication in a CANopen system

– *Emergency object (EMCY)*, triggering interrupt-type notifications whenever device errors are detected

All the aforementioned objects are stored in a centralized repository, called Object Dictionary (OD), which holds all network-accessible data and is unique for every device. Commonly used to describe the behavior of a device, it supports up to 65536 objects. The COBs are spread to distinct areas, defining communication, device and manufacturer specific parameters. The latter are left empty and are used by manufacturers, in order to provide their own device functionalities.

The OD entries are described by electronically readable file formats, such that they are uniformly interpreted by configuration tools and monitors. According to the DS-306 standard [10] they are provided by the INI format files and termed as Electronic Data Sheet (EDS) files. These files provide a generic description of a device type. However, since CANopen allows parametrization according to manufacturer specifications, a specific file format exists and is defined as Device Configuration File (DCF). This file describes the configuration for a specific device. Nevertheless, EDS and DCF files have limitations on the validation and presentation of the data as well as require a specific editor. Therefore, new XML-based device descriptions were introduced according to the DS-311 standard [11]. These substitute the EDS with the XML Device Description (XDD) file format and the DCF with the XML Device Configuration (XDC) file format. Currently, the protocol supports both device descriptions.

2.2.1 Process Data Objects (PDO)

The real-time data-oriented communication follows the producer/consumer model. It is used for the transmission of small amount of time critical data. PDOs can transfer up to 8 bytes (64 bits) of data per frame and are divided in two types: The transmit PDO (TPDO) denoting data transmission and the receive PDO (RPDO) denoting data reception. Therefore, a TPDO transmitted from a CANopen device is received as an RPDO in another device (Figure 3). Additionally, the supported scheduling modes are:

- *Event driven*, where the transmission is asynchronous and triggered by the occurrence of an object-specific event
- *Time driven*, where transmission is triggered periodically by an elapsed timer
- *Synchronous transmission*, triggered by the reception of the SYNC object, further divided in:
  - Periodic transmission within an OD-defined window (synchronous window), termed as *Cyclic PDO* transmission

  - Aperiodic transmission according to an application specific event, termed as *Acyclic PDO* transmission

- *Individual polling*, triggered by the reception of a remote request (see [9])

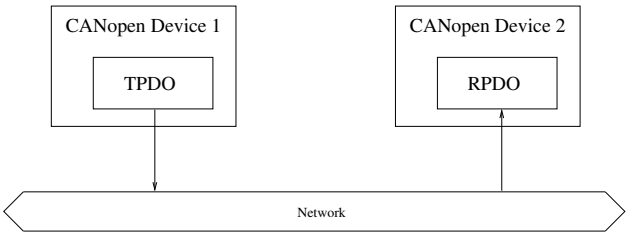


Figure 3: PDO communication

Each PDO is described by two OD sub-objects: The Communication Parameter and Mapping Parameter. For a TPDO (OD entry 6144 and 6656 accordingly) the former indicates the way it is transmitted in the network and the latter the location of the OD entry/entries, which are mapped in the payload. On the contrary for a RPDO (OD entry 5120 and 5632 accordingly) the former indicates how it is received from the network and the latter the decoding of the received payload and the OD entry/entries where the data is stored.

The Communication Parameter entry includes the *Communication Identifier (COB-ID)* of the specific PDO, the scheduling method, termed as *transmission type*, the *inhibit time* and the *event timer*. The inhibit time (expressed as a multiple of 100 μs) defines the shortest and the event timer (expressed as a multiple of 1 ms) the longest time duration between two consecutive transmissions of the same PDO.

The Mapping Parameter describes the structure of a PDO. It can be of two types, that is, static or dynamic. Static mapping in a device cannot be changed, whereas dynamic mapping can be configured at all times through an SDO.

2.2.2 Service Data Objects (SDO)

The service oriented communication follows the client/server model. It supports large, non-critical data transfers and uses three modes to allow peer-to-peer asynchronous communication through the use of virtual channels:

- *Expedited transfer*, where service data up to 4 bytes are transmitted in a single request/response pair.
- *Segmented transfer*, where service data are transmitted in a variable number of request/response pairs, termed as segments. In particular it consists of an initiation request/response followed by 8-byte request-response segments.
- *Block transfer*, optionally used for the transmission of large amounts of data as a sequence of blocks, where each one contains up to 127 segments.

A CANopen device can either receive or request an SDO, therefore these objects are not separated as the PDOs, instead they are distinguished according to their identifiers (Section 2.2.4). The communication is always initiated by a device defined as client in the network towards the server, nonetheless information is exchanged bidirectionally with two services: *Download* and *Upload*. The former is used when the client is attempting service data transmission to the server, whereas the latter when it is requesting data from the server. In both services the use of the virtual channel ensures that the received SDO is identical to the transmitted (Figure 4), unlike in PDO communication. Each request or receive SDO uses byte 0 as metadata, containing important information about the transmitted object and reducing the payload to seven bytes per frame. This byte includes the command specifier, which indicates the type of the frame, that is initiate or domain segment and request or response. The command specifier is either termed as client command specifier (*ccs*) for the client device or server command specifier (*scs*) for the server device. For

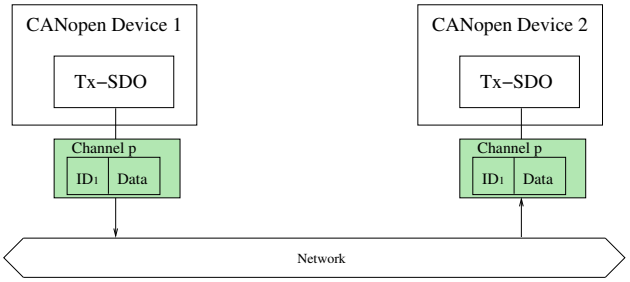


Figure 4: SDO communication

the initial request/response pair byte 0 also determines which of the three modes is used (see [13]). If transmission errors are detected either on the client or the server side, data transfer is aborted through the SDO abort frame. SDOs are used for configuration and parametrization, but also allow the transmission of a large quantity of asynchronous data, consequently they are always assigned a lower priority than PDOs.

2.2.3 Predefined objects

These specific objects provide additional functionalities to the protocol. Their transmission is following the producer/consumer communication model. Particularly, the SYNC and the TIME object are always transmitted from a specific device (Producer), according to the OD specification, whereas the EMCY object can be transmitted by any device in the network (dynamical configuration). The Predefined objects are always assigned with a high priority, in order to be transmitted as soon as possible.

The SYNC object is used to enable synchronized operation. Yet, if the transmission is handled by the CAN protocol the derived delays due to non-preemption can result to a certain jitter. Thus, if it does not provide the required accuracy for the synchronization, CANopen enables the use of the TIME object, containing a reference clock time. Though implementing a different synchronization mechanism, this object is used for accuracy, measuring the difference between theoretical and the actual transmission time of the SYNC and transmit it through a subsequent PDO.

The EMCY object is used in internal error conditions in a device and transmitted as an interrupt, in order to notify other devices. However, no notification is present when the internal error is fixed and thus the other devices cannot know the change of condition. Consequently, its implementation is not considered mandatory in CANopen systems.

2.2.4 Network configuration

Considerable complexity in CANopen systems is found in the configuration and allocation of a frame identifier to each COB. As CANopen allows parametrization the allocation scheme can be configured according to specific manufacturer requirements, however sufficient attention must be given to the priority group of each object. Therefore, to reduce the complexity of CANopen system development, a default allocation scheme based on the CAN protocol is provided, namely the Predefined Connection Set. As defined by this scheme, every object is assigned an identifier (COB-ID) according to Table 1, derived from its priority in the protocol. Nevertheless, each frame has its own identifier, since the COB-ID is augmented by the specific identifier of the node transmitting it. Every device can use up to four TPDOs, four RPDOs, one EMCY and one SDO. All the COB-IDs can be configured, except of the SDOs, if the particular device allows it.

2.2.5 CANopen system example

In Figure 5 we illustrate a simple example of CANopen system, based on the DS-401 device profile [12] and consisting of three devices. The two Slave devices are gathering analogue data from temperature sensors (Analog Temperature Detectors or ATD). The sensors are scanned periodically, whenever the event timer expires, initiating a new scan cycle. If a change in a sensor's temperature value is detected, termed

Communication Object	COB-ID
NMT	0
SYNC	128
EMCY	129
TIME	256
TPDO1	385-511
RPDO1	513-639
TPDO2	641-767
RPDO2	769-895
TPDO3	897-1023
RPDO3	1025-1151
TPDO4	1153-1279
RPDO4	1281-1407
Tx-SDO	1408-1535
Rx-SDO	1536-1663

Table 1: Predefined Connection Set

as *Change-of-State (CoS)*, an event driven PDO (configuration in Table 2 and mapping in Table 3) with the temperature value will be transmitted to the Master device, after the required conversion from the ADC. The Master device is responsible of obtaining the analogue temperature values through the DAC conversion and displaying them in an LCD display. It can also trigger the transmission of the SYNC object, informing the Slaves to abort the ongoing scan cycle and accordingly start a new one. The total number of temperature sensors in each slave device is not fixed and should be provided to the Master device during the boot-up process of the network through an SDO Download operation to its OD.

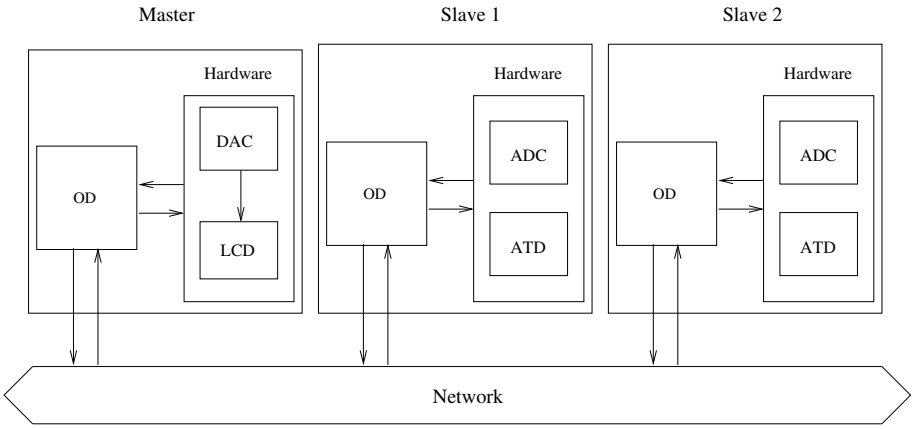


Figure 5: Example of a CANopen system

3 The BIP component framework

The BIP framework (Behavior-Interaction-Priority) [2] supports a layered component construction methodology (Figure 6, facilitating the hierarchical system composition. The lower layer (Behavior) is described by finite-state automata or Petri-Nets and models the behavior of transition systems, termed as atomic components. Each transition is labeled by an action name, termed as port, but also associated with a guard and functions manipulating a set of variables. Guards are Boolean expressions enabling conditions in the component states. The use of ports in the second layer (Interaction) defines strong or loose synchronization upon data exchange, through the use of connectors. A connector is a list of ports of atomic components which may interact. Thus, an interaction is defined as strong synchronization, when all the ports of a connector are involved (graphically represented by a bullet), whereas in the opposite case it is defined as loose

Index	Subindex	Description	Value
6144 (1800h)	0	Number of entries	5
	1	COB-ID	641 + deviceID
	2	Transmission type	255
	3	Inhibit time (in ms)	1
	4	Reserved	-
	5	Event timer (in ms)	1000

Table 2: TPDO configuration

Index	Subindex	Description	Value
6656 (1A00h)	0	Number of entries	1
	1	1st object to be mapped	25600 (6400h)/Subindex 1
25600 (6400h)	0	Number of analogue inputs	n
	1	input 1 (in °C)	30.5
::::	::::	::::	::::
	n	input n (in °C)	23

Table 3: TPDO mapping

(graphically represented by a triangle). The third layer (Priority) restricts any non-determinism between simultaneously enabled interactions. A set of atomic components can be composed into a generic compound component by the successive application of connectors and priorities.

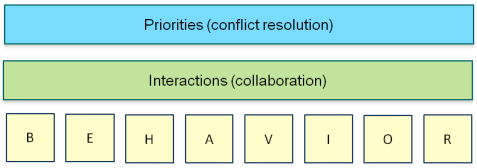


Figure 6: BIP layered component model

Figure 7 illustrates an example of a BIP composite component, comprised by two atomic components, the Sender and the Receiver. The ports *TICK*, *SEND* and *RECV* are used for the interactions between them. Each time both components are in the *idle* state the interaction involving the *SEND* and *RECV* ports is enabled. Its selection will lead to an update of variable *r*. The Sender and Receiver will respectively move to the *transmit* and the *receive* state. Consequently, both components will interact through the port *TICK* and increment variable *t*. Nevertheless, the Receiver component is also able to interact through the *EXE* port, in order to receive interruptions from other components. This conflict is resolved deterministically by priority π_1 : $TICK < EXE$, allowing the transition involving port *EXE* to be chosen, when they are both enabled. On the contrary, port *COM* of the Sender component is not enabled as long as variable *t* is less than a specific value (here 100), due to the specified guard. As an interrupt may trigger port *EXE* before this value is reached, port *TICK* is evenly enabled in the *idle* state of the Receiver component.

The BIP toolset [5] includes a rich set of tools for modeling, model transformation, analysis (both static and code generation) and execution of BIP models. It builds on a dedicated modeling language for describing BIP components. Besides purely functional primitives for describing behavior (extended automata with C/C++ functions) and composition glue (connectors and priorities), the language offers additional constructs for expressing probabilistic and/or timing constraints. The front-end BIP tools allow editing and parsing of BIP descriptions, and generating an intermediate model, followed by code generation (in C/C++) either for execution on platforms or for analysis using dedicated simulation-based tools. In particular, the BIP toolset has recently incorporated a statistical-model checking tool [4] that allows verification of probabilistic properties by combining simulation with statistical methods, thus guaranteeing the correctness and confidence of the results. Additionally, specific source-to-source transformation can be used beforehand,

[5] http://www.bip-components.com

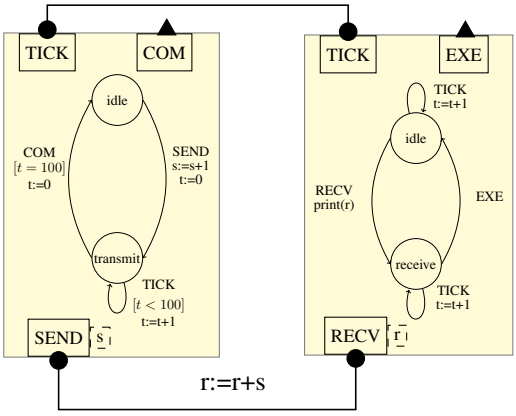


Figure 7: BIP components example

in order to increase the performance of such analysis tools.

4 Modeling CANopen in BIP

CANopen systems in BIP consist of two communication layers. The top (application) layer components represent CANopen devices, responsible for the frame transmission or reception, respectively called Device components. Following the CANopen specification, the model defines always a Device component as the Master, responsible for the transmission of the SYNC object and all the remaining Devices as Slaves. For the bottom (network) layer, we consider that communication is handled by the Controller Area Network (CAN) protocol. The CANopen Device component interacts with the CAN protocol, in order transmit or receive frames through the Bus. Therefore, prior to the derived structural translation of the CANopen primitives and communication mechanisms in BIP, we provide a brief introduction to the CAN protocol model [20].

4.1 CAN protocol model

In our previous work [20] we developed a BIP model for the classic CAN as well as the newly developed CAN FD protocol [6]. The construction of the protocol model is using a library of CAN components, which in turn allows modularity and reusability. The soundness of the model was proved by the application in benchmark automotive systems, indicating similar results with RTaW-Sim [21].

The supported version of the protocol is the Basic CAN [22], meaning a single transmit and a single receive buffer are used for the transmission and the reception of the frames accordingly. The model is also compliant with the High Speed physical layer standard [22], due its higher baud rate and interoperability with the higher-level protocols mentioned in Section 1. Finally, the current version is not modeling transmission errors.

The BIP model uses two generic types of components: the CAN station and the CAN bus. The former represents the hardware transceivers and the acceptance filters of the protocol and serves as an intermediary, in order to transmit frame requests generated from the upper layer to the Bus or equally deliver the received frames from the Bus to the upper layer. Thus, it is modeled as a compound component consisting of the Controller and the Filter atomic components accordingly. The latter represents the Bus functionality, preserving entirely its arbitration and broadcast mechanisms. Data transmission is synchronous, that is, all stations receive synchronously the frames sent by any of them. Furthermore, the underlying communication is a two-step process: first data are transmitted to the CAN bus and consequently broadcasted to all the CAN stations, including the sender. The transmission of each CAN frame field is followed by strong synchronization between the CAN stations and the CAN bus, through the use of interactions between the ports.

The CAN protocol model architecture is presented in Figure 8. It uses two groups of ports for its interactions, consisting of:

1. The *REQUEST* port (frame transmission) and the *RECV* port (frame reception) used for the interactions with the upper layer.

2. The *SOF*, *ARBITRATION*, *CONTROL*, *DATA*, *ACK*, *EOF* ports used for the interactions between the CAN station and the CAN bus component

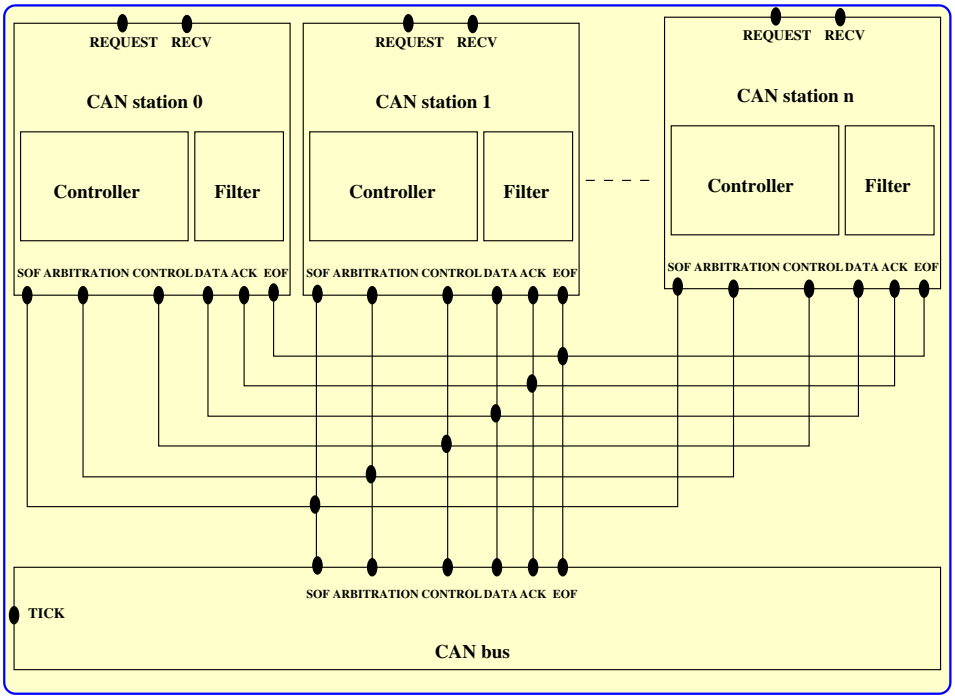


Figure 8: Generic model of a CAN system

The main types of components used in a CAN systems are:the CAN station and the CAN bus. CAN stations mediate the frame transmission on the CAN bus. They are later connected to application components. The CAN bus is modeling the arbitration and the broadcast mechanisms of every frame to all the connected CAN stations. The frame transmission process consists of two steps. First data are sent to the CAN bus and then broadcasted to all the stations, including the sender. Strong synchronization between all the CAN stations and the CAN bus is used for the transmission of each frame field. Each frame sent over the CAN bus can be of two types: data transmission (data frame) or data request (remote frame). In both cases it is represented by the tuple id, rtr, ide, length, payload, whose meaning is as follows:

- id is the frame identifier
- rtr is the Remote Transfer Request (RTR) bit
- ide is the IDentifier Extension (IDE) bit
- length contains the length of the data to be sent
- payload contains the data

For CAN higher-layer protocols (such as CANopen) rtr and ide are automatically recessive (binary 1) for every frame.

4.1.1 BIP model of CAN stations

CAN stations are composite components consisting of two atomic components: the CAN Controller and the CAN Filter. These components are responsible for the frame transmission to the CAN bus (*REQUEST* interaction) and the frame transmission to the application (*RECV* interaction) accordingly. The Controller component uses a transmission queue, in order to store the pending frames, that is, received from the application and waiting to be sent over the Bus. The queuing policy can either be of type First-in-First-Out (FIFO) or High Priority First (HPF), where frames are selected according to their priority. The selection is application-specific.

The Controller component (Figure 9) is modeled as a Petri-Net, which (1) receives frames from the application and (2) sends or receives frames from the CAN bus component. The transmission process is initiated through the *REQUEST* port, which stores the received frame in the transmission queue. If the Controller has a frame to send, the transmission cycle begins (*SOF* port). Next, in the arbitration phase, labeled by the *ARBITRATION* port in the model, every Controller sends its identifier (*id* field) to the CAN bus. The minimum identifier wins the arbitration and gets broadcasted to all of them [6]. The Controller with the minimum identifier is allowed to proceed with the transmission of the length and payload fields, while all the others are receiving them. The end of the transmission cycle is denoted by the EOF port. Throughout this cycles duration the *REQUEST* port is always available, ensuring the seamless frame reception from the application. If at least one receiving CAN station receives the frame fields correctly, the sending Controller will stop retransmitting its frame. The receiving Controllers send the frames to the Filter component through the port *RECV*.

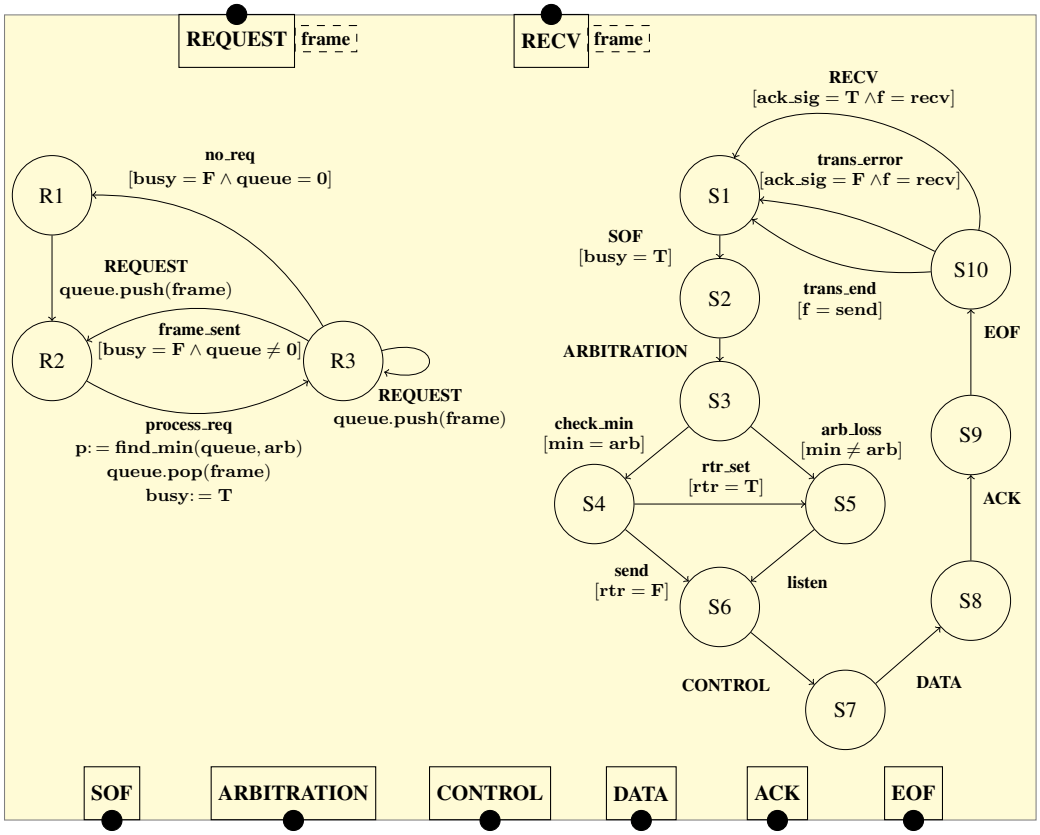


Figure 9: CAN Controller component

The acceptance filters receive every frame from the Bus, in order to either deliver it to the application or ignore it. Thus, the Filter component is receiving all the frames through an interaction involving its

[6] If a Controller has no frame to send its identifier will be automatically set to 2^{11} for the standard frame and 2^{29} for the extended

HANDLE port and the *RECV* port of the Controller component, such that only the needed frames are delivered to the application. It checks accordingly their dentifier to a list of identifiers provided by the application (*check*). If the identifier belongs to the list, the frame is transmitted through the transition *RECV*, otherwise it is discarded through the transition *filtered*.

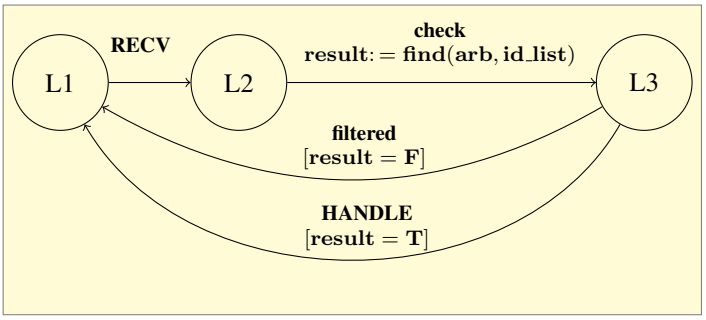


Figure 10: CAN Filter component

4.1.2 BIP model of the CAN bus

The CAN bus component is using two groups of ports for its interactions:

- The *TICK* port denoting one time step advance and,
- The *SOF*, *ARBITRATION*, *CONTROL*, *DATA*, *ACK*, *EOF* ports used for interaction with the Controller component

As shown in Figure 11 the CAN bus is receiving the frame fields *id, rtr, ide, length, payload* from the Controller component. A significant difference to the Controller is the modeling of the discrete time step advance needed for the transmission of each frame field, denoted by the port *TICK* in the model. One tick corresponds to the time needed for the transmission of one bit (τ_{bit}). The role of the CAN bus is to synchronize all the CAN stations. During the transmission cycle it interacts with all the CAN station components through the *SOF* port. The identity of the data frame sent to the Bus is determined through a check on the ide field, providing information about the number of bits transmitted through the *ARBITRATION* port. The resulting value is 12 for a standard frame and 32 for an extended representing the time needed for the arbitration phase, accordingly stored in variable g. The time duration for the transmission of the payload field (*DATA* port) will depend on the value of the length field received through the *CONTROL* port (6-tick time duration). The checksum computation results in a 16-tick time duration. The transmission cycle ends through the *EOF* port, which along with the *ACK* port correspond to a 9-tick time duration. The presence of the Interframe space (IFS) between consecutive frame transmissions is used to avoid Bus overload occurrences and corresponds to a 3-tick time duration in the model. After this time elapses the control returns to its initial state (*ifs* port). The overall frame transmission time in the model is given by:

$$C_{frame} = (32 + g + 8 \times length)\,\tau_{bit} \tag{1}$$

However the bit-stuffing encoding technique (Section 2) may increase the aforementioned time by:

$$C_{stuffing} = \left\lfloor (23 + w + 8 \times length - 1)\frac{s}{100} \right\rfloor \tau_{bit} \tag{2}$$

where $w = g - 1$, since the remote request bit is not subject to stuffing, $\tau_{bit} = 1$ and $s \in [1, 25]$ is a parameter of the model, denoting the number of stuffed bits for every frame. If the frame payload is known beforehand, this number is calculated directly from the sequence of transmitted bits, whereas in the opposite case it can be rather chosen from a probabilistic distribution provided as an input to the model. Related to the analysis provided in [15], our model is not considering the IFS field as part of the frame and

the worst-case transmission time is provided with *s* equal to 25. In every case, the number stuffed bits is denoted by the variable *stuff* in the model and added to the transmission time after the DATA interaction (Figure 11). Related to the analysis provided in [15], our model is not considering the IFS field as part of the frame and the worst-case transmission time is provided with *s* equal to 25.

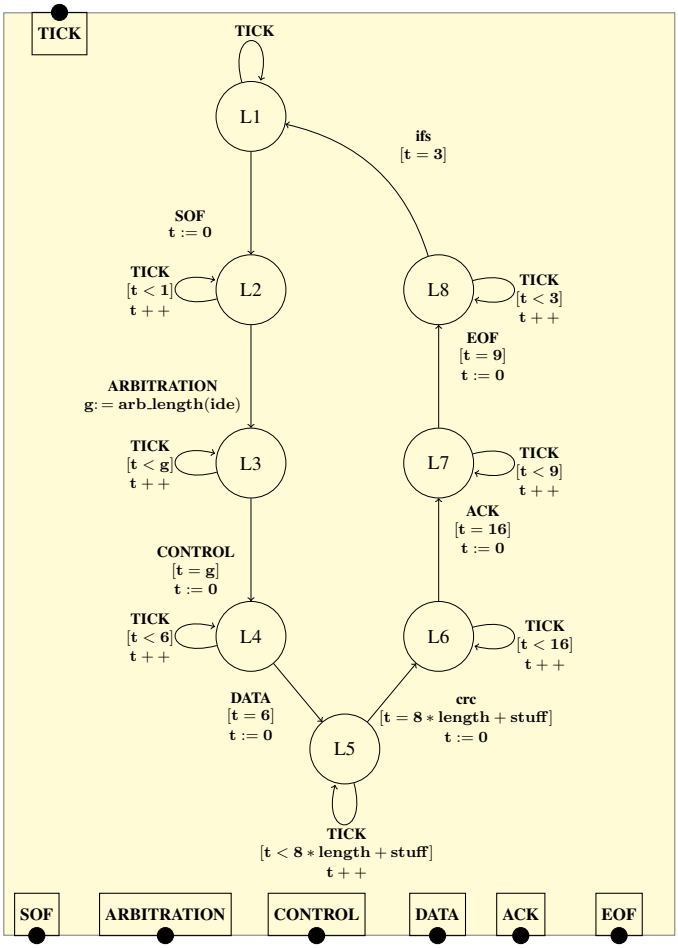


Figure 11: CAN bus component

However, the overall frame transmission time is not completely derived from Equation 2, due to the additional queue-waiting time for every generated frame, termed as blocking time. This additional time depends on the choice of the queuing policy for each CAN station component, and on the selection or not of transmission offsets as well as of abortable or non-abortable transmission requests [19].

4.2 CANopen model

The modeling of CANopen systems in BIP is structural. Every Device component is composed from several sub-components, corresponding to COBs present in the device OD. As illustrated in Figure 12, the generic CANopen Device component is composed of three parts: a transmitting part (TRANSMIT), a receiving part (RECEIVE) and a third part involving both transmission and reception (TR). Each part consists of a set of components, implementing the protocol's communication mechanisms. Each component is directly derived from a COB of the device OD, such that it will belong to one of the main categories mentioned in Section 2. In particular, PDO components can either exist exclusively only in the TRANSMIT or the RECEIVE part, or they can also be unused for the specific Device, meaning that they will not exist in any part. The same policy applies to SDO components with the difference that if they exist for the specific Device, they are included in the TR part. Furthermore, only one of the dashed SDO components is allowed to operate

in the system at a time, thus the interactions between them are not maximal (loose synchronization). In the Predefined objects component category though only one Device can exist in the transmitting part and all the other on the receiving, meaning that they are exclusive for every Device. Therefore, only one of the dashed SYNC objects will be associated with the Device of Figure 12.

Each component is responsible for the handling of a COB as a frame and consists of the tuple: *(id,length,payload)*, where *id* is the value of the COB-ID for a particular frame. In the model it belongs to the Predefined Connection Set (Section 2.2.3). Thereafter, *length* contains the length of data and *payload* the actual data of the frame.

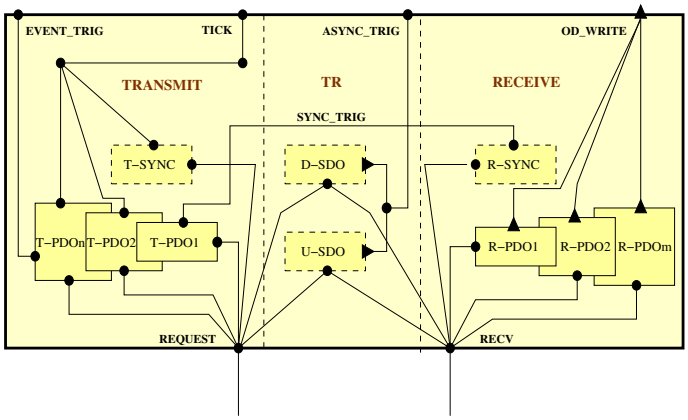


Figure 12: Generic CANopen Device component

We accordingly detail the behavior of the generic components used in the model, according to the COB category they belong. Each component, except the ones belonging to the SDO category, is atomic and described by abbreviations. These denote the part it belongs and the name of the object derived from, i.e SYNC Transmitter (T-SYNC) or SYNC Receiver (R-SYNC).

The generic CANopen Device component consists of four groups of ports using strong or loose synchronization upon interactions:

- The first implements interactions between different CANopen objects, such as the *SYNC_TRIG* port
- The second implements interactions between the Device component and the lower communication layer, such as the *REQUEST*, *RECV* ports
- The third implements interactions between the Device component and application-specific components, such as the *EVENT_TRIG*, *ASYNC_TRIG*, *OD_WRITE* ports
- The fourth implements specific interactions for general synchronization and invokes all the previously listed groups, such as the *TICK* port

4.2.1 Process Data Objects (PDO)

The PDO component types implement all the supported scheduling policies, as illustrated in Section 2.2.1. Consequently they can be of three types: SYNC-triggered, time-triggered and event-triggered. Each type is further divided in two categories: T-PDO and R-PDO. The PDO component types implement all the supported scheduling policies, as illustrated in Section 2.2.1. Consequently they can be of three types: SYNC-triggered, time-triggered and event-triggered. Each type is further divided in two categories: T-PDO and R-PDO.

Each T-PDO component is responsible for the correct initialization and generation a TPDO (*REQUEST* port). In particular, the SYNC-triggered T-PDO component following the interaction between its *SYNC_TRIG* port and the R-SYNC component (Section 4.2.3), generates a synchronous PDO or performs another device-specific action. Evenly triggered by external interrupts is the event-triggered T-PDO

component, through the port *EVENT_TRIG*. Finally, the time-triggered component implements a specific timer modeling the time step advance, through the *TICK* port. When this timer expires a time driven PDO is generated. Figure 13 presents the SYNC-triggered T-PDO component, responsible for the transmission of a TPDO2 frame, when it is triggered by a SYNC frame. It consists of the states *trigger*, *transmit* and the ports: *TICK*, *SYNC_TRIG* and *REQUEST*, also corresponding to transition labels. A connector between the *SYNC_TRIG* ports of this component and the component used for the reception of the SYNC (Section 4.2.3) ensures a synchronized operation, such that the T-PDO component moves from the *trigger* to the *transmit* state. The PDO parameters have to be provided before the transmission is triggered through the *REQUEST* port. After the interaction with the lower communication layer, it returns to the *trigger* state.

The corresponding RPDO components are responsible for the reception of a specific COB frame, provided as a parameter. They are triggered by lower-layer frame receptions (*RECV* port) and subsequently check the id of the received frame. If it is the expected frame its payload is written to the OD of the receiving Device component, through the port *OD_WRITE*. The particular OD entry is provided by the Mapping Parameter corresponding to the specified COB. This process may accordingly trigger a device-specific action. A particular example the component associated with the reception of TPDO2 frames is R-TPDO2 (Figure 13). Since, it is a receiver component it consists of the states *idle*, *receive* and the ports: *SYNC_TRIG* and *RECV*. It is also triggered by lower-layer frame receptions (*RECV* port) moving to the *receive* state, where it checks the id of the received frame. If it is the TPDO2 frame sent by the aforementioned T-PDO component, the frame's corresponding payload will be written to the OD of the receiving Device component, through the port *OD_WRITE*. This process may accordingly trigger a device-specific action.

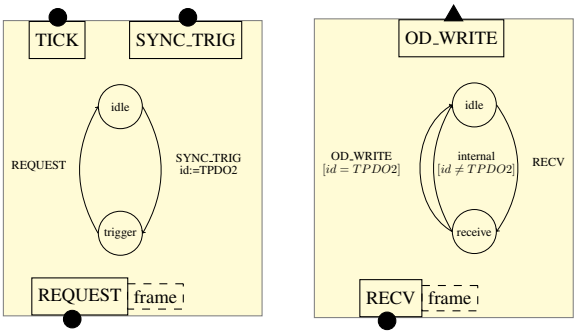


Figure 13: T-PDO and R-PDO components

4.2.2 Service Data Objects (SDO)

The SDO components are of two types: SDO Download (D-SDO) and SDO Upload (U-SDO) according to the protocol's communication mechanisms. The D-SDO and U-SDO components are responsible for configuration data exchange in the model, using one of the mechanisms presented in Section 2.2.2. They correspond accordingly to the SDO Download mechanism and the SDO Upload mechanism. As the SDO frames are accosiated with two COB-IDs (Section 2.2.4), the Device transmitting the actual data is associated with the Tx-SDO COB-ID, whereas the Device receiving them with the Rx-SDO COB-ID. The D-SDO and U-SDO are implemented as compound components in the model, consisting of a Client and a Server atomic component. The former is illustrated in Figure 14. The SDO components do not implement any timing model, since service data transmission in CANopen is asynchronous. The Client component is always initiating data transmission, after it is triggered by an external event, through the *ASYNC_TRIG* port. The D-SDO Client component is presented in Figure 15. Apart from the *ASYNC_TRIG* port it interacts with the *REQUEST* and *RECV* ports, used for interactions with the lower communication layer. All its remaining ports are internal. Initially, in the *S1* state it moves to the *S2*, whenever it is triggered by an asynchronous event. Accordingly, it determines if service data transmission is expedited or segmented. After the data request (*REQUEST* port) it remains in the *S3* state, until it receives (*RECV* port) a frame whose id is *1408+clientID* and the received server command specifier (scs) is valid. *ClientID* is the iden-

tifier of the specific client device. If the transmission was expedited (bit *e* of byte 0 is set) it will return to the initial state (S1), otherwise it will repeat the aforementioned process for all the subsequent segments, initialized according to the device OD and denoted in the model by variable *N* (model parameter). The variable counter is decremented in every successful transmission of a request/receive pair, until it is equal to 1, indicating the last segment (bit *c* of byte 0 is set). Afterwards, the component moves to the initial state, otherwise it proceeds to the next segment by the transition *next_segment*. The *toggle* variable is used to identify the sequence of successfully received request/response segments (bit *t* from payload byte 0).

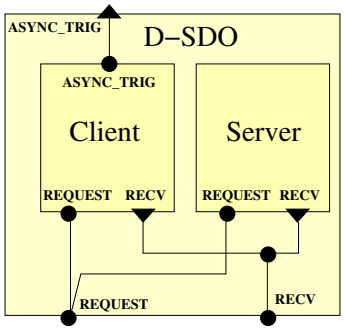


Figure 14: D-SDO compound component

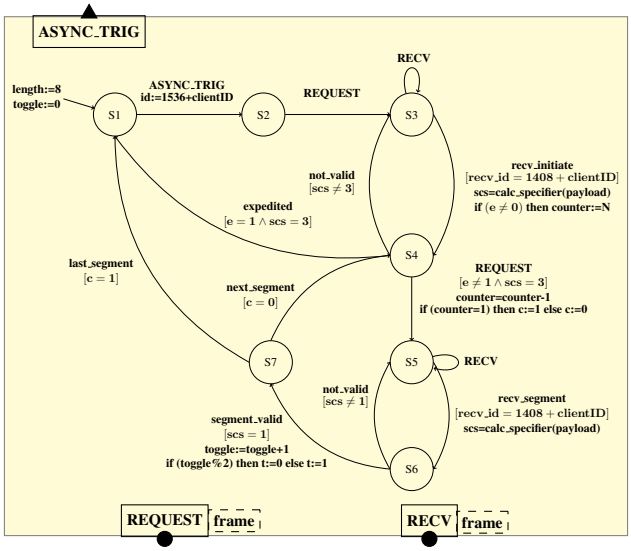


Figure 15: D-SDO Client component

4.2.3 Predefined objects

This category is focused on the SYNC object, as the other objects are not considered mandatory according to Section 2.2.3.

As for PDO, the SYNC components are divided in two categories: T-SYNC and R-SYNC (Figure 16). The T-SYNC component is responsible for the SYNC frame transmission. It consists of the states *idle*, *transmit* and the ports: *TICK* and *REQUEST*. Initially it is in the *idle* state, where it interacts through the *TICK* port. This port denotes the notion of step time advance in the model, which is calculated and stored in the variable *t*. When *t* is equal to the value of the SYNC period, defined in the device OD, the transmission is triggered by an internal move to the state *transmit*. The transmitted *frame* is initialized with the SYNC object parameters before the transmission through the port *REQUEST*. Subsequently, the

component moves to the trigger state. The R-SYNC component is controlling the SYNC-triggered PDO transmission. It only triggers a frame transmission upon the successful reception of the SYNC frame. This component consists of the states *idle*, *receive* and the ports: *SYNC_TRIG* and *RECV*. When a frame is received through the *RECV* port, used for the interactions with the lower communication layer, it will move to the *receive* state. It returns to the *idle* state either by triggering the transmission of a PDO frame (*SYNC_TRIG* port), or internally. The choice is controlled by a specific guard.

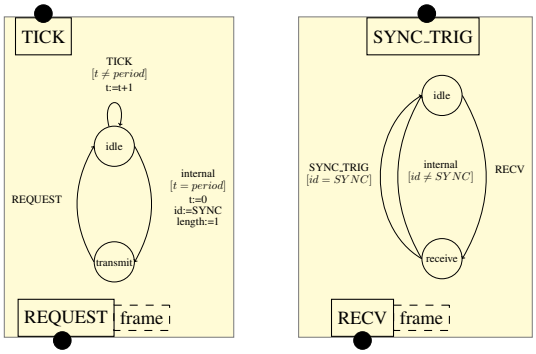


Figure 16: T-SYNC and R-SYNC components

4.2.4 Timing and version issues

A constraint that has to be carefully considered in our model is the choice of the time step advance for the *TICK* interaction. Its granularity has to be relative with the baud-rate (speed) of the CAN protocol. Therefore we consider the time needed for the transmission of one bit to the Bus equal to one-step advance in our model. For example a baud-rate of 500 kbit/s, corresponds to a time step advance of 2 microseconds (μs). Subsequently, $2\mu s$ of real time will be taken as a one-step advance in our model.

The version of the CANopen protocol model represents the functionality of the most recent communication profile [13], which additionally implies that the SYNC object is not anymore mapped to an empty frame, but includes an 1-byte counter as payload. Moreover, currently we don't consider hardware or transmission errors. Therefore, the SDO abort frame is not included in the model.

4.2.5 Concluding remarks on the modeling

The construction of a formal model facilitated the identification of some important issues in CANopen communication. Initially, in SDO communication the data size parameter is optional and usually not indicated in CANopen systems before as well as during the transfer. Even though this type of objects should always be addressed with the lowest priority, the receiver cannot perform a consistency check, which is consequently reducing the robustness of the protocol. Another important issue is related to the number of unused data bytes in some SDO frames, instead filled with padding, in order to follow the 7-byte data request/receive pair specification. The outcome is the introduction of significant overhead to the lower-layer transmission protocol, which might cause additional delays in the transmission of high-priority frames, especially during SDO block transfers.

Overall, the built component libraries contained 14 types of atomic components for CANopen and 3 types of atomic components for CAN.

5 Validation and experiments

The conducted experiments are focusing on validation and performance evaluation of CAN and CANopen applying the presented approach to two case studies. The first concerns a deterministic powertrain network benchmark [14], triggering periodic or stochastic data transmission through the CAN bus. The second is related to the Pixel Detector Control System (DCS), used as the innermost part for the ongoing ATLAS

experiment at CERN's Large Hadron Collider (LHC) particle accelerator [3], where communication is handled by CANopen.

5.1 Powertrain network

5.1.1 Deterministic model

For this case study we compare our approach with existing domain-specific tools, such as NETCAR-BENCH [8]. The comparison is done in terms of accuracy and simulation time.

The CAN application layer is represented as a collection of Device components. These components are atomic and contain a transmission and a reception part. Figure 17 illustrates the former part. Frame transmission is handled by the *REQUEST* port, whereas frame reception by the *RECV* port. Each frame is triggered when its specific period is reached (port *generate*). This is achieved by consecutively incrementing variable t whenever the interaction through the port *TICK* is possible. Specifically, this interaction is enabled until t is equal to the minimum period of the array P, responsible of storing the periods for all the frames. The size of P is a model parameter, denoted as N. As the periods here are fixed, the minimum period of every Device component is only calculated in the initial state.

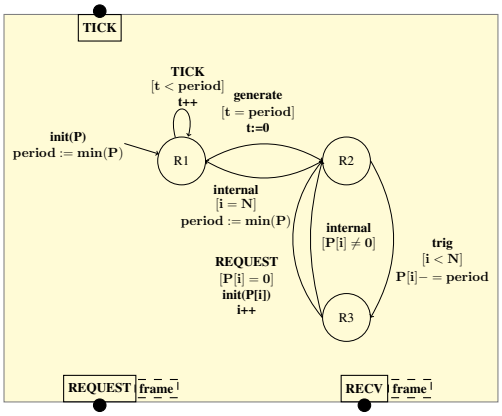


Figure 17: Deterministic powertrain component

The deterministic powertrain network benchmark was generated by NETCARBENCH. It consisted of 5 Electronic Control Units (ECUs) communicating over a Bus with a bit-rate of 500kbit/s. The queuing policy used was HPF and the observed Bus load was 13.8%, distributed approximately equal in every ECU. Bit-stuffing was fixed to 10%, meaning s was equal to 10 for every frame in Equation 2. Transmission offsets and clock drifts were not considered in this model. All parameters concerning the frame identifier, period and payload are provided in Table 4. Our analysis focused on the frame response times using two methods. The first method applied the BIP design flow on the generated benchmark, to construct and analyze the BIP system model. The derived translation represented the entire SW/HW system, reflected by the benchmark. The obtained system model was accordingly simulated using the associated simulation-based tools. The second method provided the generated benchmark as input to RTaW-Sim [21].

The system model in BIP contains 15 atomic components for the CAN protocol model and 5 atomic components for the application model. It also uses 60 connectors (40 for the CAN protocol and 20 for the application model). The total number of transitions in the system is 255 (210 for the CAN protocol and 45 for the application model). Overall the model totals about 1250 lines of BIP textual code.

Figure 18 illustrates the results obtained using the two methods, where the analysis focused in three categories, that is minimum, average and worst-case frame response times. The results were identical for both methods, in all the aforementioned categories. From the conducted analysis we can also note that approximately 55% of the frames had a deterministic response time, where the remaining 45% had a fixed blocking time, due to higher priority frame transmission.

ECU	CAN ID	Period (ms)	Payload
1	189	10	5
	200	20	1
	269	50	2
	298	50	8
	533	100	6
	685	2000	8
2	328	20	6
	371	100	8
	379	20	8
	477	50	5
	506	200	8
3	262	20	7
	427	50	7
	472	100	6
	492	100	7
	774	2000	8
	977	1000	8
4	159	20	6
	208	20	7
	321	50	7
	480	50	8
	502	100	4
	628	200	7
	690	2000	8
	776	1000	8
5	260	20	4
	307	50	6
	370	100	5
	473	50	6
	724	200	7

Table 4: Network configuration parameters

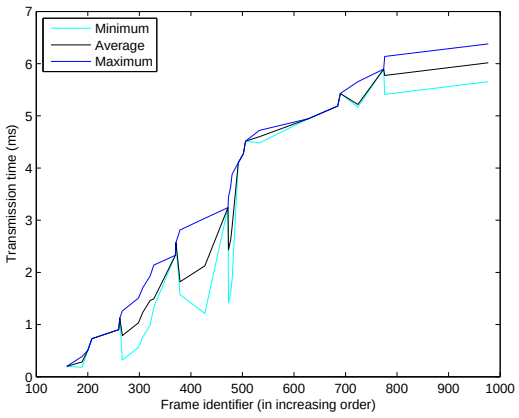


Figure 18: BIP/RTaW-Sim frame response times for the powertrain network

A real system time of 1 hour was simulated in 5 minutes and 30 seconds using the BIP simulator and in 13.5 seconds using the RTaW-Sim simulator. The observed divergence occurred due to the difference in the simulation models. The BIP simulator is state-based, whereas RTaW-Sim is an event-based simulator. Nevertheless, we are currently introducing existing model transformations [7] in the BIP system model, in order to improve the simulation time.

5.1.2 Stochastic model

We accordingly introduce a stochastic behavior to the previous powertrain network case-study. This is accomplished by adding first a probabilistic margin (jitter) for every period, subsequently reducing the load on the Bus. Each margin follows a Poisson distribution based on a mean rate equal to 1/10 of each period. The probabilistic margins are stored in the array *m* (Figure 19). These are added to the frame periods and the resulting period is stored in the array *D*. As the minimum period of every stochastic Device component is not fixed it has to be calculated iteratively and not only in the initial state. The second step is the introduction of a stochastic bit-stuffing, thus the additional response time is not fixed as well. Consequently, parameter *s* in Equation 2 varies according to a uniform distribution in the range [1,25] and each transmitted frame has a different response time. The results, shown in Figure 20, are also divided in the three aforementioned categories. As it is observed, in average all the frames have a very small blocking time. However, due to the non-deterministic behavior of the system, response times cannot be described only through the previous timing analysis. Therefore, in Figure 21 we focus on a particular frame, in order to show the probabilistic variation of the obtained response times.

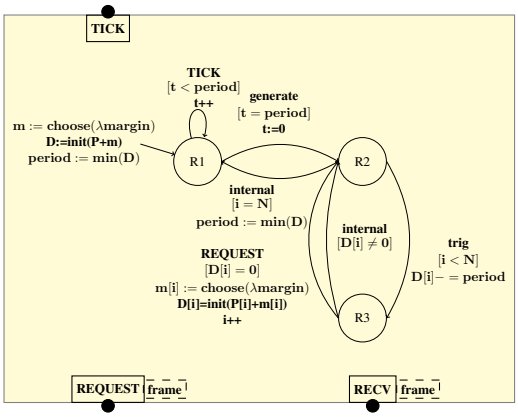


Figure 19: Stochastic powertrain Device component

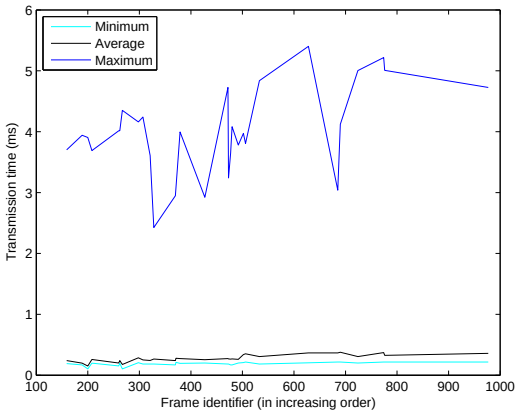


Figure 20: BIP frame response times for the stochastic powertrain network

This model exceeds the simulation capabilities of NETCARBENCH. It can also be analyzed using the recently incorporated statistical-model checking tool of the BIP toolset [4].

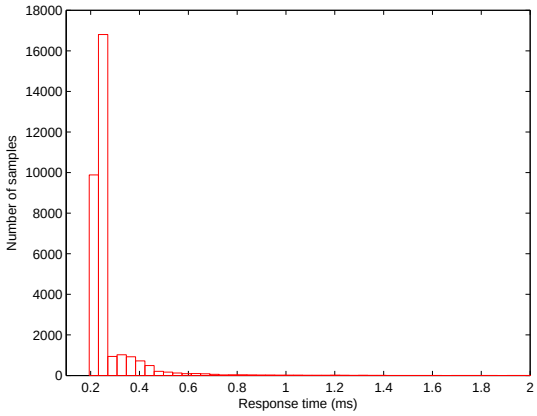


Figure 21: Response time distribution of a frame

5.2 Pixel Detector Control System

The conducted experiments focused on the Pixel Detector Control System (DCS), used as the innermost part for the ongoing ATLAS experiment at CERN's Large Hadron Collider (LHC) particle accelerator. For the particular case study we consider an extension to the test beam of 2002, previously presented in [18], used for the calibration and performance evaluation of the detector modules used in the experiment.

The chosen test beam is presented in Figure 22 and consists of two Detector systems, where each one contains four pixel detector modules. Each pixel detector module is equipped with a temperature sensor, used in order to measure its operating temperature and accordingly determine its lifetime. The measurements are subsequently provided as input to a thermal interlock system (*Interlock Box*) and a plug-on I/O board manufactured in CERN, named as *ELMB* (Embedded Local Monitor Board), in order to be transmitted to a Detector Control System (DCS) Station through the CAN Bus, using CANopen as the communication protocol. The application software as well as the hardware configuration for the ELMB board can be found in [17]. This manual also provides a full listing of the Object Dictionary, defining not only the standard objects according to the DS-401 Device Profile [12], but also manufacturer-specific objects for the ELMB.

A new scan cycle begins every 1 second and in the course of it all the pixel detector modules are scanned. A TPDO2 frame is transmitted whenever a CoS in a module is detected. The transmitted frame contains the ADC readout in counts (ADC resolution). However, after a power-up or a reset of the ELMB the ADC voltage ranges need to be re-calibrated through a TPDO3 frame. This frame contains the input voltage in μV and is transmitted prior to the generation of a TPDO2 frame. Since each temperature sensor is exposed to safety risks, the Interlock Box is responsible of comparing the input data to a reference value (threshold) as well as for the generation of a logical signal, if the temperature is found higher. The output of every Interlock Box module is provided to a Logic Unit, which is also monitored by an ELMB module. This module is used to transmit the generated signal as a TPDO1 frame, informing the associated pixel detector that it is overheated, in order to enable its Cooling Box. The coolant flow inside each Cooling Box is set and controlled by an expedited SDO frame. Therefore, two additional ELMB modules are considered, each one obtaining coolant flow data from a Regulator module. Subsequently they establish a peer-to-peer communication channel with the corresponding ELMB of each Detector module and transmit the data through an SDO Download operation. Finally, although the DCS Station is mainly used for data logging, it is also responsible for the periodical transmission of the SYNC frame, informing the ELMB module of every Detector to abort the current scan cycle and accordingly start a new one.

The bit-rate of the CAN Bus for the particular test beam is set to 125kbit/s. An equally important remark is that during the initialization phase of the system the DCS Station initializes properly, all the ELMB devices, storing via an SDO Download operation all the COB-IDs correctly in their OD.

The existence of certain requirements for the DCS ensure the proper functionality of the system. They

Index	Subindex	Description	Type
4096 (1000h)		device type	Unsigned32
4097 (1001h)		error register	Unsigned8
::::		::::	::::
5120 (1400h)		1st receive PDO communication parameter	PDOComPar
	0	Number of entries	Unsigned8
	1	COB-ID used by PDO	Unsigned32
	2	transmission type	Unsigned8
	3	inhibit time	Unsigned16
	4	Reserved	
	5	Event timer	Unsigned8
::::		::::	::::
5632 (1600h)		1st receive PDO mapping parameter	PDOMapping
	0	Number of mapped objects in PDO	Unsigned8
	1	1st object to be mapped	Unsigned32
	2	2nd object to be mapped	Unsigned32
	3	3rd object to be mapped	Unsigned32
::::		::::	::::
6144 (1800h)		1st transmit PDO communication parameter	PDOComPar
	0	Number of entries	Unsigned8
	1	COB-ID used by PDO	Unsigned32
	2	transmission type	Unsigned8
	3	inhibit time	Unsigned16
	4	Reserved	
	5	Event timer	Unsigned8
::::		::::	::::
6656 (1A00h)		1st transmit PDO mapping parameter	PDOMapping
	0	Number of mapped objects in PDO	Unsigned8
	1	1st object to be mapped	Unsigned32
	2	2nd object to be mapped	Unsigned32
	3	3rd object to be mapped	Unsigned32
::::		::::	::::
24576 (6000h)		digital input	
	0	Number of digital inputs	Unsigned8
	1	read 8 inputs 1-8	Unsigned8
25600 (6400h)		analog input	
	0	Number of analogue inputs	Unsigned8
	1	input 1	Integer16
	2	input 2	Integer16
	3	input 3	Integer16
	4	input 4	Integer16

Table 5: Object Dictionary of the ELMB

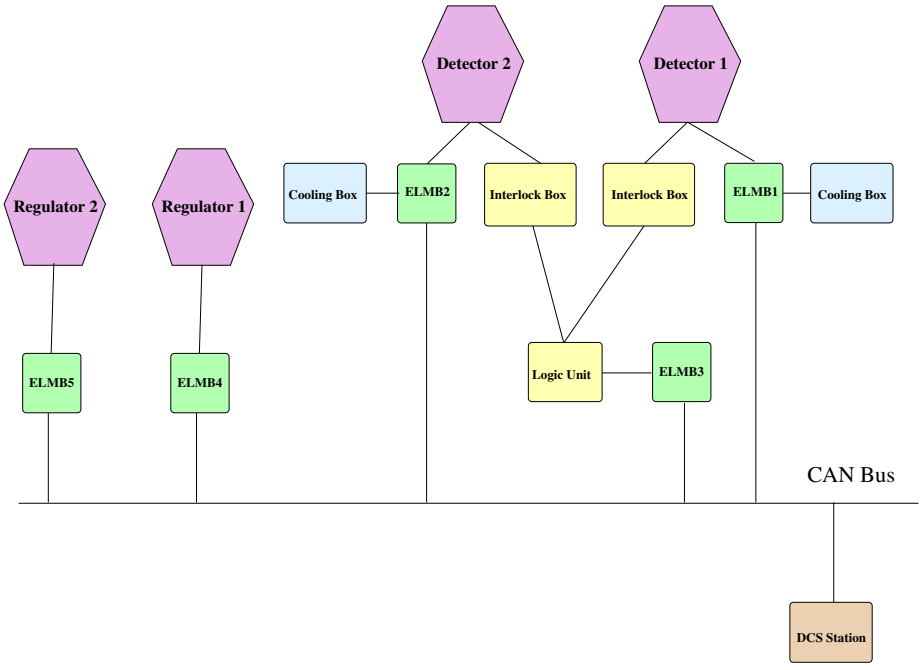


Figure 22: Pixel Detector Control System

are divided in two categories: those concerning the physics and performance of the DCS individual subsys-

tems, found in the CERN Document Server [7], and those related to the communication through CANopen. Specific requirements belonging to the second category are:

1. In case of an increased sensor temperature the Logic Unit must inform the DCS Station rapidly, through a TPDO1 frame. Thus, the TPDO1 frame must have a zero blocking time, once triggered.
2. When ELBM1 or ELMB2 is reset a TPDO3 frame should be transmitted before a CoS in a pixel detector module is detected, since the generated TPDO2 will require an ADC conversion.
3. The coolant flow must be set at least once before a Cooling Box is required to cool an indicated pixel detector module. Consequently, ELMB4 and ELMB5 should initiate the transmission of the D-SDO frame before any other frame in the network is triggered.

We constructed the model of the Pixel Detector Control System, using the library of CANopen components presented in Section 4.2. The resulting BIP system for the DCS is illustrated in Figure 23. It is comprised by 39 atomic components forming the CANopen communication layer and 13 atomic components for the CAN protocol. The generated BIP model used 95 connectors (53 for the CANopen and 42 for the lower-layer communication model). The total number of transitions for this system was 427 (174 for the CANopen and 252 for the lower-layer communication model). Overall the model totals about 2300 lines of BIP textual code.

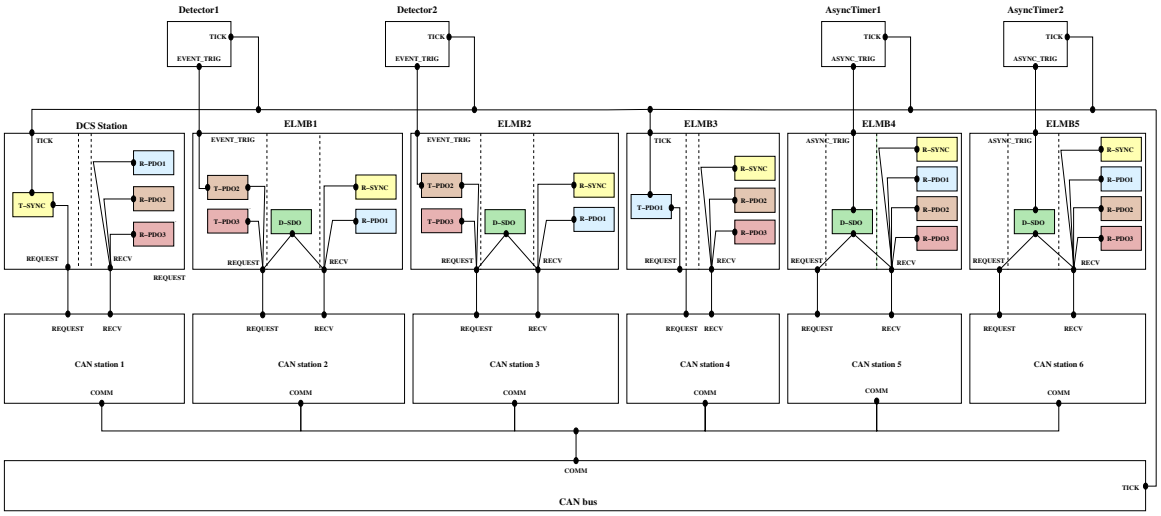


Figure 23: BIP model of the Pixel Detector Control System

For the conducted experiments we used real temperature data provided as input to the model, thus deriving a distribution for the temperature changes, as well as the reference value (threshold) for the Interlock Box. Moreover, the external event triggering an SDO transmission was modeled as an asynchronous timer generating event interrupts, whenever it expired.

A real system time of 4 hours was simulated in 2 minutes and 43 seconds using the BIP simulator. The obtained results are illustrated in terms of minimum, average and worst-case frame response times in Figure 24. The existing COB frames of the DCS system are represented in the horizontal axis. As it is observed the response times (in milliseconds) are highly dependent from the choice of the lower-layer scheduling policy (here HPF). Due to the stochastic behavior of the system, the blocking time for each transmission varies according to the Bus load at the given instant. This variation between the minimum (zero) and the worst-case (maximum) blocking time depends on the frame identifier, defining its priority on the system. In particular, the SYNC frame (COB-ID 128) has a relatively small variation compared to the D-SDO frames of ELMB1 and ELMB2 (COB-IDs 1540 and 1541 respectively). In this analysis the response time of the SDO frames is measured from the instantiation of the request frame until the transmission end of the response frame.

[7] http://cds.cern.ch/record/391176

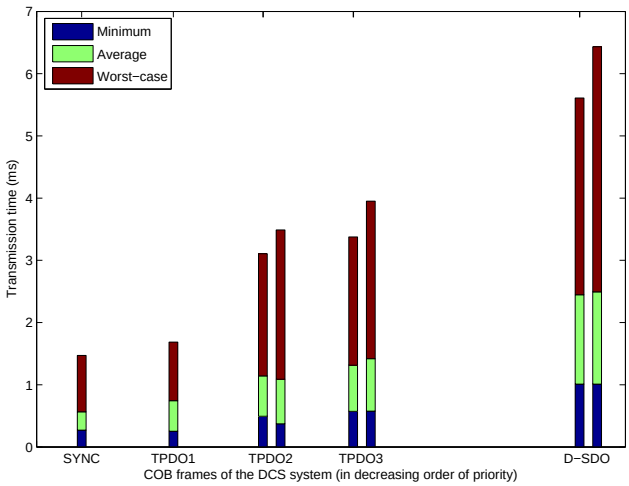


Figure 24: Frame response times computed from the BIP model

In order to evaluate the system requirements we describe the above requirements with stochastic temporal properties using the Probabilistic Bounded Linear Temporal Logic (PBLTL) formalism [4]. We accordingly present the results derived properties after an extensive number of simulations using the SBIP model checker [8].

Property 1: Requirement 1. This property is expressed as $\phi_1 = \Box^{10000000}(T_{inhibit} > T_{TPDO1})$, where 10000000 indicates the number of steps for each simulation, corresponding to a large number of communication cycles. Furthermore, $T_{inhibit}$ is the inhibit time and T_{TPDO1} the response time of the TPDO1 frame (COB-ID 388). For the DCS system $T_{inhibit}$ is equal to 1 sec, which much greater than the worst-case response time of TPDO1 ($T_{TPDO1_{max}} = 1.72$ msec from Figure 24). Therefore $P(\phi_1) = 1$ and this requirement is always satisfied.

Property 2: Requirement 2. In the second experiment, we try to estimate the property $\phi_2 = \Diamond^{10000000}(T_{TPDO2} < T_{TPDO3})$, where 10000000 is explained as above, T_{TPDO2} and T_{TPDO3} denote the response time of TPDO2 and TPDO3 following an ELMB reset (Figure 25). The conducted experiments have shown that if a scan cycle is initiated through the reception of the SYNC frame, a CoS can be detected before the generation of a TPDO3 frame. However, a reset in ELMB1 or ELMB2 occurred in approximately 3% of the simulations, thus this property was quantified as $P(\phi_2) = 0.005$. This probability is equal to the tool's level of confidence, thus the requirement is considered as satisfied.

Property 3: Requirement 3. Finally, we tested through simulation the property: $\phi_3 = \Box^{24000}(t_{TPDO2} > t_{D-SDO})$, where 24000 is the number of steps required for the initialization period as T_{init} is 2 sec, t_{TPDO2} is the system time at the end of the TPDO2 frame transmission and t_{D-SDO} is the system time at the beginning of the D-SDO frame transmission. Since the D-SDO frame was generated asynchronously, this property was quantified as $P(\phi_3) = 0.1$. As it is observed by Figure 26, focusing in a specific simulation, the TPDO2 frames from ELMB1 and ELBM2 finish their transmission before a D-SDO frame. Moreover the conducted experiments have shown that even when the D-SDO frame was generated before the first instance of a TPDO2 frame, it was mostly blocked due to its lowest priority for this system.

[8] http://www-verimag.imag.fr/Statistical-Model-Checking.html

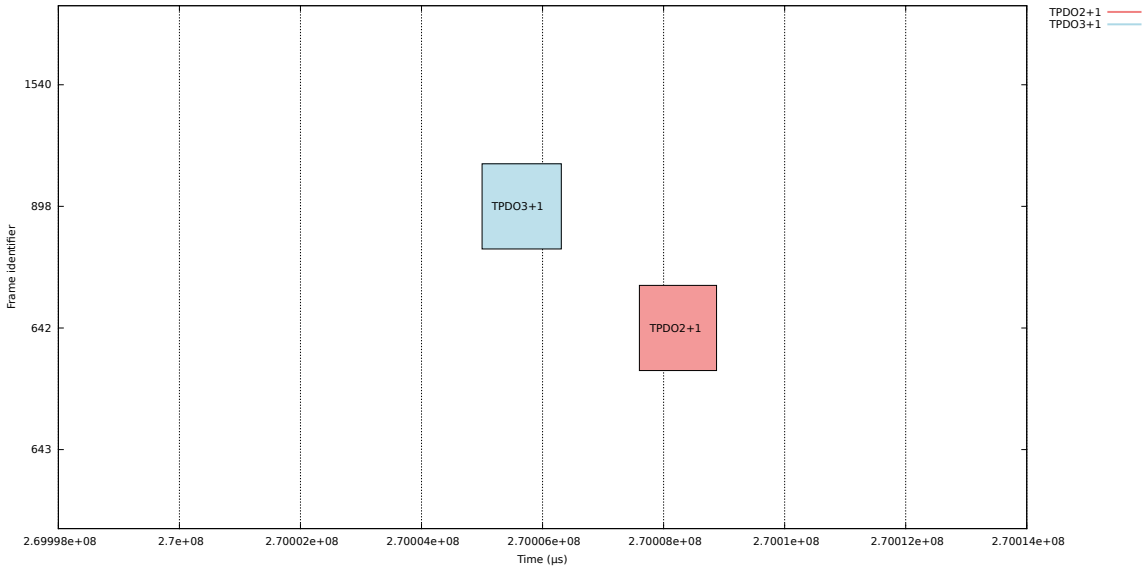


Figure 25: Response time graph following a reset in ELMB1

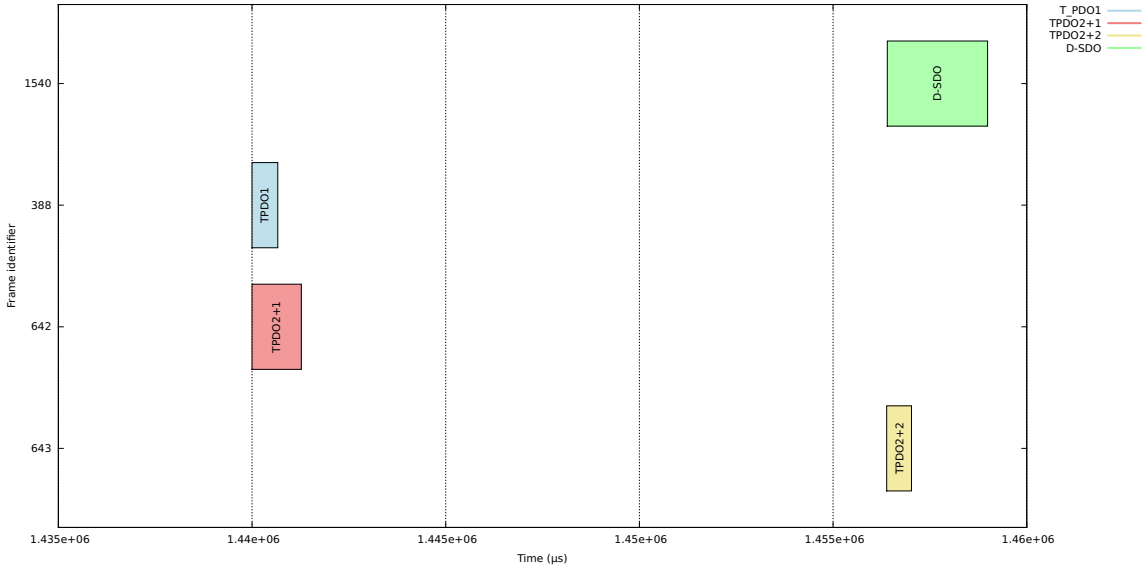


Figure 26: Response time graph for TPDO1, TPDO2 and D-SDO

6 Conclusion and ongoing work

We have presented a systematic method to construct detailed functional models for CANopen systems in the BIP component framework. The construction method is fully structural, that is, it preserves the CANopen system structure in BIP, meaning systems consisting of a number of network-connected devices, which in turn, consist of a number of interacting communication objects, as defined in the device and the communication profile accordingly. The model captures both functional and extra-functional aspects, referring to timing characteristics for periodic and aperiodic transmission. The models are fully operational, they can be tested, simulated and validated using statistical model checking tools available in the BIP toolset.

For the time being, our model uses the CAN protocol for the low-layer network communication. How-

ever, its use also raises practical limitations as with unconfirmed services, since there is no possibility of knowing when a frame is lost. A possible solution would be the use of individual polling from the Master device. Nevertheless, this method will produce additional overhead, if there is no CoS in a number of devices. Furthermore, the two COB-IDs defined for SDO communication allow only one client/server channel in the network at a time. In the opposite case collisions or conflicts are inevitable. Additionally, the growing use of extensive networks and the rising data load are increasing the complexity of CANopen systems nowadays. One of the main reasons behind this is the low bandwidth (1 Mbit/s) and the limitation in the network length (127 nodes and 25m max bus length). CAN FD was introduced, in order to ameliorate the former limitation, nonetheless the bandwidth is only increased during the data transmission period.

For all these reasons, we are working on further extensions, in order to support CANopen systems deployed on other wired or wireless protocols. The two most interesting protocols in this domain are the IEEE 802.3, used for wired Ethernet communication and the IEEE 802.11, used for wireless communication. In the scope of these extensions, we shall map the presented primitives and communication mechanisms of CANopen to each aforementioned protocol.

References

[1] M Barbosa, M Farsi, C Allen, and AS Carvalho. Formal validation of the CANopen communication protocol. In *Fieldbus Systems and Their Applications 2003:(FET 2003): a Proceeedings Volume from the 5th IFAC International Conference, Aveiro, Portugal, 7-9 July 2003*, volume 5, pages 226–238. Elsevier Science Limited, IFAC, July 2003. 1

[2] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, pages 3–12. IEEE, 2006. 1, 3

[3] C Bayer, S Berry, P Bonneau, M Bosteels, G Hallewell, M Imhäuser, S Kersten, P Kind, M Pimenta dos Santos, and V Vacek. Studies for a detector control system for the ATLAS pixel detector. In *Proceedings of the 7th Workshop on Electronics for LHC Experiments (LEB 2001)*, page 396, 2001. 5

[4] Saddek Bensalem, Marius Bozga, Benoit Delahaye, Cyrille Jegourel, Axel Legay, and Ayoub Nouri. Statistical Model Checking QoS properties of Systems with SBIP. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 327–341. Springer, 2012. 3, 5.1.2, 5.2

[5] Robert Bosch. CAN specification version 2.0. *Robert Bosch GmbH, Stuttgart*, 1991. 1, 2.1

[6] Robert Bosch. CAN with Flexible Data-Rate specification. *Robert Bosch GmbH, Stuttgart*, 2012. http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf. 4.1

[7] Marius Bozga, Mohamad Jaber, and Joseph Sifakis. Source-to-source architecture transformation for performance optimization in bip. *Industrial Informatics, IEEE Transactions on*, 6(4):708–718, 2010. 5.1.1

[8] Christelle Braun, Lionel Havet, Nicolas Navet, et al. NETCARBENCH: a benchmark for techniques and tools used in the design of automotive communication systems. In *7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems-FeT'2007*, pages 321–328, 2007. 5.1.1

[9] CAN in Automation. *Application Note 802*, August 2005. 2.2.1

[10] CAN in Automation. Electronic data sheet specification for CANopen, Draft Standard 306, 2005. 2.2

[11] CAN in Automation. CANopen device description, Draft Standard 311, 2007. 2.2

[12] CAN in Automation. CANopen Device Profile for Generic I/O Modules, Draft Standard 401, June 2008. 2.2.5, 5.2

[13] CAN in Automation. Application layer and communication profile, Draft Standard 301, February 2011. 1, 2.1, 2.2, 2.2.2, 4.2.4

[14] Jeffrey A Cook, Jing Sun, Julia H Buckland, Ilya V Kolmanovsky, Huei Peng, and Jessy W Grizzle. Automotive powertrain control: A survey. *Asian Journal of Control*, 8(3):237–260, 2006. 5

[15] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007. 4.1.2

[16] Embedded Systems Academy. *CANopen Magic User Manual*. http://www.esacademy.org/products/getfile.php?filename=COMPDLLManual.pdf. 1

[17] Henk Boterenbrood. *CANopen application firmware for the ELMB*, November 2011. http://www.nikhef.nl/pub/departments/ct/po/html/ELMB128/ELMB24.pdf. 5.2

[18] S Kersten, KH Becks, M Imhäuser, P Kind, P Mättig, and J Schultes. Towards a Detector Control System for the ATLAS Pixel Detector, 2002. 5.2

[19] Dawood Ashraf Khan, Robert I Davis, and Nicolas Navet. Schedulability analysis of CAN with non-abortable transmission requests. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference*, pages 1–8. IEEE, 2011. 4.1.2

[20] Alexios Lekidis, Marius Bozga, Didier Mauuary, and Saddek Bensalem. A model-based design flow for CAN-based systems. In *14th International CAN Conference, Eurosites République, Paris*, 2013. 4, 4.1

[21] Nicolas Navet, Aurélien Monot, Jörn Migge, et al. Frame latency evaluation: when simulation and analysis alone are not enough. In *8th IEEE International Workshop on Factory Communication Systems (WFCS2010), Industry Day*, 2010. 1, 4.1, 5.1.1

[22] Olaf Pfeiffer, Andrew Ayre, and Christian Keydel. *Embedded networking with CAN and CANopen*. Copperhill Media, 2008. 2.2, 4.1

[23] port GmbH. *youCAN CANopen prototyping*. http://www.port.de/fileadmin/user_upload/Dateien_IST_fuer_Migration/youCAN_e.pdf. 1

[24] Thilo Schumann. CANopen Conformance Test. In *Fieldbus Technology*, pages 152–156. Springer, 1999. 1

[25] DeviceNet Specification. Release 2.0, including Errata 4. *April*, 1:1995–2001, 2001. 2.1

[26] SAE Standard. J1939: Recommended practice for a serial control and communication vehicle network. *J1939*, 201308, 2013. 2.1

[27] Vector Informatik GmbH. *CANanalyzer User Manual*. http://vector.com/vi_manuals_en.html. 1