A Model-based Certification Framework for the EnergyBus Standard

Alexander Graf-Brill[1], Holger Hermanns[1], and Hubert Garavel[2,3,4]

[1] Saarland University — Computer Science
66123 Saarbrücken, Germany
{grafbrill,hermanns}@cs.uni-saarland.de
http://depend.cs.uni-saarland.de
[2] Inria
[3] Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
[4] CNRS, LIG, F-38000 Grenoble, France
hubert.garavel@inria.fr
http://convecs.inria.fr

Abstract. The EnergyBus is an upcoming industrial standard for electric power transmission and management, based on the CANopen field bus. This paper reviews the particularities of the EnergyBus architecture and reports on the application of formal methods and protocol engineering tools to build a model-based conformance testing framework that is considered to become part of the certification process for EnergyBus-compliant products.

1 Introduction

Light Electric Vehicles (LEVs) are booming in many countries: In the Netherlands for instance, a traditionally bike-affine country, the last year has seen LEV sales outnumber ordinary bike sales with respect to total revenue. Many different OEMs and unit suppliers, including for instance Bosch and Panasonic, but also fleet operators such as Deutsche Bahn are active in this new market, and the annual market growth, especially for *pedelecs* – pedal assisted electric vehicles – is predicted to be at least 20% for the coming years. Bike vendors throughout Europe are feeling this trend and they react by turning into LEV vendors. When doing so, one of the first problems they face is the multitude of plugs and sockets in use to connect battery packs and chargers. Different OEMs partly use the same plug type, but with different pin interpretation. Cellphone users may know this problem, but in the LEV context this problem is safety critical, because the battery capacities are much larger, and charging them in the wrong way may make them catch fire.

The EnergyBus association[5] is a consortium assembling all major industrial players (and Saarland University); their intention is not limited to interoperability between chargers and batteries, but broader, namely to ensure interoperability between all electric LEV components so that one eventually can freely

[5] http://www.energybus.org

combine battery pack, motor, charger, and even the dashboard, as long as all devices are EnergyBus-compliant. At the core of this initiative is a universal plug integrating a CAN-Bus with switchable power lines. The initiative has caught further momentum by broadening its scope to stationary smart power micro-grids [1], making it a full-fledged standardisation effort for *Energy Management Systems* (EMS).

To make the devices interoperate requires however more than standardised plugs and sockets. A protocol stack is needed, which orchestrates the exchange of messages and guarantees safe interoperation. LEV and CAN-bus experts are progressing fast on the design and standardisation of this protocol stack; version 1.0 of the EnergyBus standard was released on March 2011 and updated with version 1.0.6 in August 2012 [2]. Version 2.0 is about to be published and serves as the starting point for the recent work of the IEC/ISO/TC69/JPT61851-3 standardisation commision. This work raises many challenging issues: Are these protocols correct? Do their implementations (originating from multiple device manufacturers) conform to the EnergyBus standard?

The authors had the unique opportunity to closely follow and interact with the standardisation activities, applying state-of-the-art protocol engineering methods and tools to the EnergyBus specifications under design. To formally describe the EnergyBus protocol, we used the *LOTOS New Technology* (LNT) language [3], a successor of the LOTOS [4] and E-LOTOS [5] ISO/IEC international standards for the formal specification of communication protocols and distributed systems. We used the LOTOS and LNT compilers and verification tools of the CADP toolbox[6] [6]. Based on the TGV [7] model-based test generator, we developed a tool platform for the automatised conformance testing [8] of EnergyBus/CANopen implementations against the formal specification. The platform is ready to be used as a mandatory step in a certification process which is to be rolled out by the EnergyBus association as soon as first prototype devices become available seeking the EnergyBus-compliance label. This can be a door opener for formal methods research in a much broader context than model-based testing, and in an industrial area with rapidly growing societal and economic impact. To prepare for that, this paper gives a detailed and precise account of the EnergyBus architecture, together with a discussion of the formal modelling and testing activities performed.

The paper is organised as follows. Section 2 briefly recalls the principles of the CANopen field bus and Section 3 presents the main features of the EnergyBus. Section 4 reports about the formal specification work done for the EnergyBus and Section 5 describes the model-based framework set up to check the conformance of EnergyBus implementations. Section 6 discusses related work and Section 7 concludes the paper and draws perspectives about future work.

[6] `http://cadp.inria.fr`

2 CANopen

The communication backbone of the EnergyBus is the CANopen field bus protocol [9]. The latter was developed from 1993 to 1995 in the ESPRIT project ASPIC under chairmanship of Bosch and is a widely adopted field bus protocol in the automation area. The standard is administrated by the "CAN in Automation" (CiA) association[7], which manages and supports standardisation of CAN-related applications. CANopen in turn is based on the *Controller Area Network* (CAN) bus protocol but it is designed in such a way that other protocols can replace the two lowest (in the sense of the ISO/OSI model) layers.

CANopen, in its original form, is a very robust Carrier Sensing Media Access protocol with Bit Arbitration (CSMA/BA). In the CAN philosophy, every communication task has its own dedicated message name — the function code — which is combined with the node-id to form the *Communication Object Identifier* (COB-id).

The network itself is built up by several nodes sharing a CAN line, which is accessed in an equal bit-rate setting and with uniquely assigned node-ids. CANopen adds several protocols for different communication tasks and a local interface object for on-top applications — the *Object Dictionary* (OD). This is basically a two-dimensional array structure with predefined interpretations of the different entries. The directory entries are used to exchange data values between the application and the CANopen network, and to store configuration parameters for the various CANopen protocols.

Some CANopen protocols are relevant for the discussion that follows. The *Network Management* (NMT) protocol enables a dedicated master node to manage the basic operation status of other nodes in the network: such "slave" nodes can be started, stopped, reset, or brought to operational state.

The *Service Data Object* (SDO) protocol is mainly used for setting up and configuring the devices inside a network. This is a client/server protocol between two nodes. Each two nodes use a fixed pair of COB-ids for their communication, a so-called SDO Channel.

Contrary to SDO, the role of the *Process Data Object* (PDO) protocol is the periodic transmission of status information or application data, and event notification.

Error control service is supported by the *Heartbeat protocol*, in which each node periodically sends its current NMT state to the network. A time-out of these messages is used to signal the "loss" of the node to every listening node.

Emergency messages (EMCY) are used to report the occurrences of internal device errors to the network participants.

The so-called *Layers Setting Services* (LSS) [10] provide basic configuration mechanisms to assign node-ids and to adjust the communication baud rate of single network nodes. These services are orthogonal to the other CANopen protocols and are especially applied to unconfigured devices, for which identification is derived from the device unique "virtual type" label.

[7] http://www.can-cia.org

3 EnergyBus

The EnergyBus Standard [2] aims at a common standard for electric devices in the context of Energy Management Systems (EMS). This includes the definition of a connector family, on the one hand, and appropriate communication protocols, on the other hand. The central and innovative role of the EnergyBus is the transmission and management of electrical power. So the purpose of its protocol suite is not just to transmit data, but in particular to manage the safe electricity access and distribution inside an EnergyBus network.

The development of the EnergyBus protocols is in the hands of the EnergyBus e.V. and its members. Conceptually, it extends the underlying CANopen architecture with several components, and the EnergyBus protocols are developed in terms of *CANopen application profiles* endorsed by the CiA association [2]. Among these, the 'Pedelec Profile 1" (PP1) is very detailed and targets a predominant business context.

3.1 Power Lines and EnergyBus Devices

An EnergyBus network contains one or several power lines, which may be of two kinds: the *main power lines*, which carry either DC (ranging from 12 to 250 V) or AC (from 85 to 265 V), and the *auxiliary power line*, which carries low-voltage DC (between 9 and 12 V) and is always powered. The type of the network can be restricted by application profiles, such as the PP1, which has a single main power line and an auxiliary power line (12 V).

CANopen devices are physical entities implementing CANopen specifications. EnergyBus devices are CANopen devices that also implement EnergyBus specifications. *Active devices* are connected to the main power lines, while *passive devices* are only connected to auxiliary power lines. Since malfunctioning of the main lines can be lethal, the proper functioning of active devices is crucial for electrical safety. Yet, passive devices are important too — even if they are low powered, and even if low- and high-voltage lines are strictly separated — because passive devices behaving incorrectly may interfere with active ones and put the network at risk.

3.2 Virtual Devices

The EnergyBus specification adopts the concept of a Virtual Device from the CANopen standard. Each EnergyBus device supports one Object Dictionary, several communication services, and can implement several Virtual Devices. The roles of almost all functional elements operating in an EnergyBus network are defined in terms of Virtual Devices.

A Virtual Device is usually characterised by: (1) its behaviour, usually specified as a combination of textual definitions and *Finite-State Automata* (FSAs); (2) by dedicated OD entries, namely, electric or physical parameters possibly influencing the flow of energy; and (3) by its communication settings, such as PDO definitions [11]. Each instance of a Virtual Device is assigned a set of these

PDOs together with distinct COB-ids to uniquely identify a PDO's sender. Virtual devices include:

- *General Application Objects*, which store the list of all implemented Virtual Devices, their nominal voltage and current ranges, their status and control word entries, temperatures, and so on. Each EnergyBus device must implement this Virtual Device exactly once.
- *Battery Packs* are the simplest kind of power supply in the EnergyBus. In general, they are accumulators and so have the capability of being recharged inside the EnergyBus. Moreover, Battery Packs can be removed from the network.
- *Voltage Converters* change the voltage of electrical power sources. There are different kinds of Voltage Converters offering different operation modi, the most prominent one being — at least in the LEV and pedelec setting — the *External Charger Station*.
- *Motor Control Unit* (MCU) are Virtual Devices specifying specific status values and control capabilities of motor equipments and providing protocol interfaces to them.
- The *Human Machine Interface* (HMI) is a dashboard that displays information to the user. It can be used to start/stop and configure the network, to turn on/off the lights, to choose support profiles for the motor, etc.
- The most important Virtual Device is the *EnergyBus Controller* detailed out in the next subsection.

3.3 EnergyBus Controller

The *EnergyBus Controller* (EBC) is responsible for managing the distribution of electric power. In general, several EBCs can coexist in an EnergyBus network, but to avoid interferences, there is always exactly one EBC active, which has the fixed node-id "1", implying highest priority — all the other EBCs being basically turned off. The active EBC has the authority to turn on/off the entire EnergyBus and to control its attached devices.

The EBC is supposed to ensure electrical safety of the network, especially protection against over/under voltage or current, and achieves this by limiting the power flows according to parameters collected as characteristic and actual values from the devices attached. To do so, the EBC sets appropriate limits to other devices and dynamically adjusts these limits according to the actual settings of the system. Nevertheless, every device is ultimately responsible for its own safety and must protect itself from damage by disconnecting itself from power lines if necessary.

The EBC functionality requires the device hosting the EBC to provide the CANopen NMT master functionality to control the communication behaviour, and LSS master functionality to initialise and configure the devices as network nodes. The EBC controls the internal state and monitors the Heartbeat of every connected device. It maintains SDO channel connections to all slave devices for controllability reasons. Each Virtual Device (except the EBC itself) maintains an SDO server channel to the EBC and no client channels.

3.4 Subtask Protocols

Besides the general task to safely distribute electric power throughout the network, there are special situations in which the EnergyBus uses specific smaller protocols, e.g. for charging Battery Packs, updating the devices' software, diagnostics, energy saving via sleep mode functionality, etc.

4 Formal Modelling of the EnergyBus

A formal specification is a pre-requisite for modern protocol engineering approaches; the present section reports about the specification activities carried out for the EnergyBus.

4.1 CANopen and EnergyBus Specification Documents

As mentioned in Section 3, the EnergyBus is defined on top of CANopen, and EnergyBus devices are CANopen devices, which inherit features from CANopen and extend them with additional ones. Thus, a formal specification of the EnergyBus requires to model certain CANopen features (e.g., NMT, LSS, OD), although not in full detail.

The CANopen and EnergyBus specifications are given informally, as a combination of text, protocol flow charts, and FSAs — the latter being used to summarise the behaviour of devices as seen by other devices. Master devices are entitled to modify the current states of slave devices by sending special messages. The CANopen specifications are stable, whereas all EnergyBus protocols (except perhaps the Boot Loader protocol) have been evolving quickly.

The basic CANopen standard is defined in [9] (about 150 pages). CANopen associates to each device several data structures (e.g., the OD) and various services, among which the NMT, PDO, SDO, EMCY, and (optional) LSS. There are additional services of CANopen, each represented by its own set of protocols, being enabled and disabled according to the current state of the NMT FSAs.

Concerning the NMT: The behaviour of each NMT slave node in the network can be seen as a 6-state FSA. Textual explanations are provided for each of these states: *Initialising*, *Reset Application*, and *Reset Communication* describe initialisation stages, whereas *Pre-Operational*, *Operational*, and *Stopped* represent communication configurations. The NMT service enables the NMT master node to manipulate the NMT FSAs of the NMT slave nodes. The corresponding NMT protocols are provided in textual form and sequence diagrams.

Concerning the LSS: While being optional in CANopen, this service turns mandatory in an EnergyBus network. It is defined in [10] (about 60 pages) as an FSA with four states, and several corresponding protocols.

The EnergyBus standard is defined as a collection of 14 documents [2], 11 of which describe particular types of Virtual Devices, giving for each device its specific OD entries and specifying its behaviour as one or several FSAs. The EnergyBus standard brings various extensions to CANopen, among which the

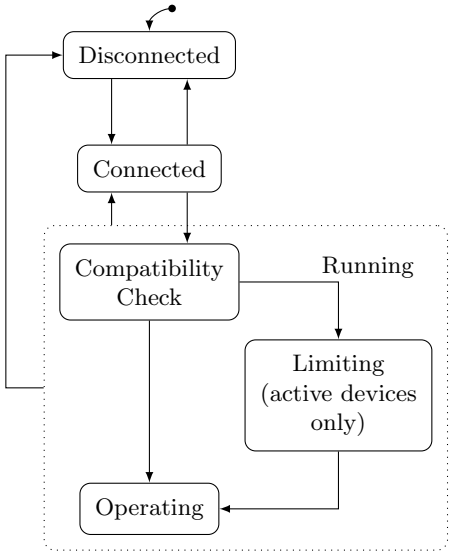

Fig. 1. EnergyBus Energy Management System FSA

Energy Management System (EMS) and its *Boot Loader* and *Sleep Mode* protocols, each being defined by a specific FSA.

Concerning the EMS: This service is at the core of the EnergyBus; the basic behaviour and interaction capabilities of an EnergyBus Device, as seen by the EBC, is represented by the EMS FSA depicted on Fig. 1. A device is in the *Disconnected* state when not connected to the EnergyBus, possibly powered by an external source. A device is in the *Connected* state when the connector is plugged, i.e., when it is connected to the CAN network and when the EnergyBus can be used as an auxiliary power supply. The three states labelled *Running* describe the successful connection to the CANopen/EnergyBus network. Devices in state *Compatibility Check* are examined to check the compatibility of their electric/electronic characteristic values to those of already connected devices. If the device is connected to the main power lines of the EnergyBus, its variable power settings are being adjusted in state *Limiting*. Finally, devices in state *Operating* can execute their running application. The EBC controls the EMS FSA of the connected EnergyBus devices via SDO writes to the *control word* entry of their Object Directories.

4.2 Formal specification of the EnergyBus in LNT

To formally model the EnergyBus, we chose the LNT specification language [3], which is the most recent descendent in the family of LOTOS [4] and E-LOTOS [5] languages. In a nutshell, LNT combines functional languages — to describe data types and functions operating on typed values — and process calculi —

to describe concurrent agents that execute in parallel, synchronise using rendezvous, and communicate via message passing. As usual with process calculi, the operational semantics of LNT is defined in terms of *Labeled Transition Systems* (LTSs) and *Structural Operational Semantics* (SOS) rules. A translator from LNT to LOTOS exists, thus enabling LNT specifications to be compiled, executed, and verified using the CADP toolbox [6].

As mentioned above, our formal specification is based on the CANopen and EnergyBus standards. For the latter, we took into account about 400 pages of documentation, out of which approximately 300 pages describe OD entries. The modelling process took six months, with several iterations to follow the EnergyBus specification updates and to introduce abstractions in the models.

Our LNT model [12, Appendix C] is divided into several parts, each representing a single aspect (e.g., a specific service) of the CANopen/EnergyBus specifications. Each part is split into a header file containing the type and channel definitions, and a file containing the process definitions. These files are assembled together by means of *cpp* preprocessor directives (e.g., `#include`, `#ifdef`, etc.) to produce a model having about 1700 lines of LNT code and 200 lines of comments. Table 1 provides data concerning the different parts, the second column providing best-effort estimations, as information about a given component being often distributed across multiple documents.

Table 1. Size of informal and formal specifications

Component	Documentation (pages)	LNT code (lines)
NMT	8	260
Heartbeat	6	200
EMCY/Error	4	145
LSS	62	360
EMS	3	440
PDO	45	60
SDO	25	30
OD/Variables	300	70

The FSAs are directly translated to LNT code by implementing their states as LNT enumerated types and their corresponding actions as LNT processes. The different FSAs are intended to run in parallel, loosely coupled to each other. They synchronise and communicate using a mixed approach that combines message passing and shared variables. There is also a master/slave hierarchy between FSAs; for instance, all services must monitor the current state of the NMT by consulting the corresponding variable's value. LNT was found sufficiently expressive to describe this particular model of coordination. Like most process calculi, LNT does not support shared variables as a primitive communication paradigm, but it is easy to model them as additional parallel processes.

A straightforward modelling of the OD in LNT would consist of a two-dimensional array structure hosting different types of data; yet, this would cer-

tainly cause an explosion in the number of states and transitions in the underlying LTS. We therefore chose a refined modelling approach by only modelling relevant OD entries, and by "specialising" our LNT code depending on the specific nature of each OD entry: (1) *global constants* common to all devices are modelled as LNT constant functions; (2) *local constants* specific to a particular device are modelled as parameters of the LNT process(es) corresponding to this device; (3) *local variables* internal to a given device are modelled as LNT local variables; and (4) *global variables* shared between several devices are modelled, as explained above, using dedicated LNT processes with communication gates to read and write these variables and, if needed, an additional channel that notifies all "listeners" of any value update.

We also specialised the translation of PDOs and SDOs into LNT, and only modelled their useful aspects, which saved much modelling effort compared to producing fully-detailed generic process models for them. Each defined PDO [11] is represented by its own LNT process, parameterised by the PDO's COB-id and the actual names of the corresponding variables. SDOs are not represented as LNT processes, but as pairs of read/write operations affecting specific variables.

4.3 Results and Discussion

As it is often the case, formal modelling revealed problems in textual specification documents. For instance, it uncovered ambiguities in a former version of the LSS on how to take the NMT initialisation process into account; also, the Sleep Mode protocol exhibited gaps [12, Appendix B] preventing its formalization. There have been several iterations with the industrial partners to solve issues raised by confusing and non-consistent naming, and mismatches between natural language and logics.

The LNT specification developed for the EnergyBus standard was mechanically checked using the LNT2LOTOS, CÆSAR.ADT, and CÆSAR compilers provided by the CADP toolbox [6]. These tools enabled to fix various mistakes, such as undefined identifiers and typing errors. This is a significant enhancement compared to most protocol standards, which are "only" checked by human reviewers.

An automated search for deadlocks was performed. More stringent analyses could have been done using the powerful verification capabilities of CADP (equivalence checking, model checking, etc.), but we decided to focus our work on conformance testing.

5 Conformance Testing for the EnergyBus

The goal of conformance testing is to examine whether the functional behaviour a given implementation — referred to as the *Implementation Under Test* (IUT) — conforms to a (hopefully formal) specification, which defines the expected correct behaviour. In the case of certification, there are usually several implementations from various manufacturers to be compared against the same specification, which

thus plays the role of an authoritative, central reference and avoids divergence between implementations.

5.1 Model-based Testing

For the EnergyBus, we used a *model-based testing* approach in which physical CANopen/EnergyBus devices are the IUTs, and the LNT specification is the reference.

In this approach, each IUT is considered as a black box (i.e., each device is only accessible via its CAN/CANopen interface, without extra means to probe its internal state). The functional behaviour of the IUT is checked by applying a suite of *test cases* derived from the LNT specification. *Test generation* (i.e., the production of test cases) is, to a large extent, automated: taking as input the LNT specification and a set of high-level scenarios (called *test purposes*) provided by a human, a large number of detailed low-level test cases are produced automatically — thus, avoiding the frequent risk of introducing mistakes in test cases. *Test execution*, i.e., the application of test cases to the IUT — namely, sending inputs to the IUT, observing the outputs of the IUT, and comparing them to the correct outputs predicted by the LNT specification — runs automatically without human intervention.

The general model-based testing framework presented in [13] and especially [8] is the theoretical basis for this work. For test generation, we chose to use the TGV tool [7], partly because it is directly applicable to LNT specifications. We now briefly mention how to instantiate the general framework in this particular case.

Concerning the *formal domain* for LNT specifications, we use *Input Output Labeled Transition Systems* (IOLTS), i.e., LTSs whose actions are divided into three classes: inputs, outputs, or internal (τ). As the LNT compiler only produces LTSs, the user must provide a criterion to distinguish between inputs and outputs.

Concerning the *test assumption*, we assume an IOLTS where in every state every input action is enabled, which exactly models the IUT behaviour. Thus, we are able to formally reason about a relation between the specification and the IUT.

The *implementation relation* presented in [8] is **i**nput/**o**utput **co**nformance (**ioco**), which requires that the implementation, after executing some trace, is allowed to perform only those output actions permitted in the "matching" states (i.e., states reached after this same trace) of the specification.

For the EnergyBus, we must use the $\mathbf{ioco}_U$ implementation relation [14], which is slightly weaker than **ioco** and ranges over so-called "underspecified" traces. The reason is that we perform black box testing of the IUT through its CAN/CANopen interface, modelled as a test context $C[_]$: thus, the IUT is a composition of the EnergyBus device with $C[_]$, while the specification is a composition of the device LNT model with $C[_]$. By using $\mathbf{ioco}_U$, we avoid forbidden behaviour due to unspecified interaction between the IUT and $C[_]$, thus preserving the *soundness* of testing.

The *test purposes* given to the TGV tool are finite IOLTSs (specified in LNT by the user and automatically translated into IOLTSs by the LNT compiler) that describe "interesting" scenarios, i.e., paths of the EnergyBus specification that shall be considered for testing. Test purposes can be seen as high-level goals that restrict the set of possible test cases.

TGV generates so-called *test graphs*, which are finite IOLTSs. In these graphs, each execution path starting from the initial state corresponds to a *test case*, i.e., a sequence of inputs sent to the IUT and outputs received from the IUT. Test graphs are prepared to accept every output, even if erroneous, from the IUT. A test case terminates when reaching a state labelled with a *test verdict*, which is either *pass* if the path is legal according to the specification, or *fail* if the last output of the path is unexpected, or *inconclusive* when the path leaves the area of the specification defined by the test purpose when generating the test case.

5.2 Test Platform Architecture

We implemented this model-based testing approach in a hardware/software platform, the architecture of which is depicted on Figure 2.

The left-hand side of the figure represents *test generation*. The two main inputs of the TGV tool are the LNT specification of EnergyBus and the test purpose(s) given in LNT. Using the LNT and LOTOS compilers of CADP, these LNT files are translated into LTSs accessible via the dedicated OPEN/CÆSAR API [15]. Additional information about the list of input and output actions enables to turn these LTSs into IOLTSs. TGV produces as output test graph(s) encoded in the BCG format of CADP.

The right-hand side of the figure represents *test execution*. It operates on a test graph generated by TGV and a physical EnergyBus device, which is accessed through its CANopen interface. The latter is provided by emtas[8], an SME developping industrial CANopen solutions. The execution engine returns whether the device complies with the test graph(s).

The test execution software (nearly 3100 lines of C code) is logically split into modules performing different tasks. The *Explorer* module uses the BCG library of CADP to traverse the test graph. In this graph, the *Partitioner* and *IO-Chooser* modules select a test case on the fly, using a uniform random generator to choose one among several enabled transitions. The *Adapter* module provides software and hardware interface based on an industrial CANopen stack. The *Instantiator* module replaces abstract data values by concrete ones in transition actions, so as to provide the *Driver* with parameters used in the *Adapter* functions. Finally, the *Driver* module connects both worlds, checks test verdicts, and emits test results.

Although TGV can also extract test cases from test graphs, we do not use this feature and stop TGV right after test graph generation, thus avoiding the non-deterministic choices between inputs and outputs made by TGV. This way, we fit better with the theory of [8] and our test cases can handle the situation —

[8] http://www.emtas.de

likely to happen in a multi-layer communication protocol — where a particular action takes non-negligible time and is interrupted by an output of the IUT.

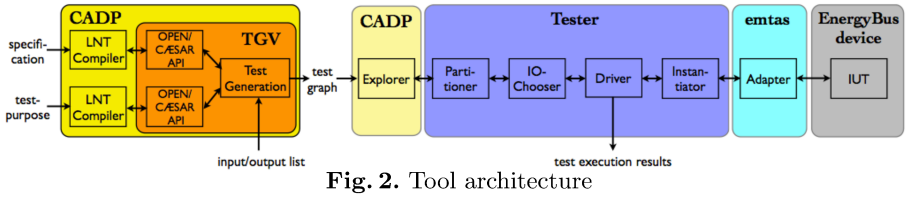

Fig. 2. Tool architecture

5.3 Abstractions

When modelling real-world examples, one often faces the state-space explosion problem. Initially, the implicit LTS representation and on-the-fly algorithms provided by CADP's OPEN/CÆSAR framework [15] (upon which TGV is built) have been sufficient to handle the complexity of our models. But adding the LSS and PDO protocols in full led to state-space explosion arising from data types, such as 8-byte message data.

To overcome the issue, we applied various abstraction techniques to appropriate areas of our model. The most effective approach was *data abstraction* [16]: data ranges for LSS addresses, and other identifying parameter values were reduced to either two-value domains (e.g., *own_id* and *wrong_id* with binary tests for equality) or three-value domains (e.g. *smaller_id*, *own_id*, and *bigger_id* with accordingly abstracted order relations). Data abstraction was also applied to energy management components. Such abstraction is perfectly compatible with the intended behaviour, which checks whether a given variable is above or below certain thresholds (absolute measurements) or compares it against its previous value (relative measurements). From a conformance testing point of view, such abstractions can be seen as a simple application of action refinement [17], transferring the complexity from the LNT model to the adapter component.

5.4 Results and Discussion

Discussions with EnergyBus developers were distilled into test purposes covering realistic scenarios, such as the initialisation of unconfigured nodes via combinations of LSS communications, or the boot-up procedure of configured nodes, including the transmission of its PDOs with dynamic and static variable values. These test purposes are of manageable size (about 80 lines of LNT code on average) and generate complex test graphs (e.g., 600 states and 1100 transitions each) that would be difficult to produce manually.

These test purposes provide the initial database for setting up a model-based certification process for EnergyBus-compliance. Since no EnergyBus devices are thus far available on the market, we emulated them with C code, using the same

software stack and hardware as the adapter component, bridged by a CAN connection. This provides us with a fully automatic way to exercise, debug and tune the testing platform itself, and to prepare for the arrival of the first prototype devices to be certified.

As our LNT specification models CANopen aspects (via the test context $C[_]$), our platform can be applied for the testing of CANopen devices as well. In fact, this enabled us to detect three deviations in the CANopen software stack [12, Appendix E] used.

1. missing counter value data in SYNC message reception;
2. missing state change in the LSS FSA during configuration process by slave devices;
3. wrong behaviour of unconfigured slaves in the LSS Fastscan protocol (not confirmed yet).

Notably, deviation (1) was not signalled by a *fail* test verdict, but by a timeout of test execution followed by a manual inspection of the test results log file. This is due to the fact that IOLTSs do not model quantitative time: the LNT specification allowed to stay in the current quiescent state, from which a sequence of internal actions could lead to the expected output; this could have been avoided in a timed setting.

6 Related work

Model-based conformance testing [13] [8] is an established technology supported by various tools, such as TGV [7] [18], TorX [19], and Uppaal-Tron [20] that generate tests for real-time systems. Model-based testing has been discussed for automotive [21] and fieldbus [22] settings, though without focussing on formal conformance testing or product certification. Formal verification was performed for TTP [23]. Deep investigations of CAN and FlexRay efficiency [24] and timing [25] have been carried out. The specific aspects of the electric vehicle domain have been detailed out in [26]. Formal models for smart energy systems have been discussed in [27]. To the best of our knowledge, the combination of formal modelling, model-based testing, and component certification has not been discussed so far in the literature, for sure not in the context of mobile energy management.

7 Conclusion and Future Work

Energy management networks are rapidly gaining importance for small- and medium-scale applications, including electric vehicles, caravans, combinations of solar panels and storage devices, etc. Our work pushes forward formal specification and model-based testing approaches via the EnergyBus standard. Harvesting a close interaction of the authors in the standardisation activities, state-of-the-art protocol engineering technology has been used to model and check EnergyBus specifications, in parallel to setting up a testing framework ready-to-use for device compliance certification. Apart from preparing for certification, our concrete contributions are threefold:

1. We produced a formal specification [12, Appendix C] in LNT of the key aspects of the EnergyBus, including the required CANopen features. This specification, which has been submitted to the EnergyBus association, provides a readable and compact model that can be used as a basis for future corporate work and automated analyses.
2. This specification work and the computer analysis of the formal specifications have revealed various ambiguities and inconsistencies [12, Appendix B] in the EnergyBus description.
3. A model-based platform for the conformance testing of EnergyBus/CANopen implementations against our formal specification has been developed. We detected three inconsistencies [12, Appendix E] in an industrial CANopen implementation; two of them have been acknowleged as defects and the confirmation for the third one is pending.

Thus, there is already a clear return on the investment in developing formal specifications. Our work shows that model-based testing approaches are applicable — if not directly out of the box, at least without too much stretching, i.e., using simple abstraction techniques. The LNT language, which had not been used priorly for field bus protocols, has shown to be a beginner-friendly notation and provided a formal basis for discussing with the EnergyBus experts. Furthermore the CADP tools (especially the LNT translator) benefited from the EnergyBus modelling feedback.

Our work is to some extent driving as well as driven by the evolution of the EnergyBus forthcoming standard. A current focus is on the subtask protocols (see Section 3.4), including the Charging Protocol. The testing platform will be connected to EnergyBus device prototypes as soon as they are available. The EnergyBus association considers a strict certification process for any device asking for the EnergyBus-compliance label: model-based testing has the potential to become a mandatory step in this process, as a supplement to conventional integration testing already used by the CiA association. To the best of our knowledge, this would then be the first time that applied formal methods are an integral and mandatory step in the introduction of a new industrial standard. It also can be seen as a spearhead for deeper modelling and analysis activities, especially with respect to real-time and power flow properties. This asks for timed and hybrid automata models, together with effective model checking techniques. As a whole, the greater LEV domain appears as a very promising arena for applied formal methods.

Acknowledgments

This work has been performed under the aegis of the Alexander-von-Humboldt foundation, partly in the framework of the German transregional DFG project AVACS[9] and of the European IST FP7 projects SENSATION[10] and MEALS[11].

[9] http://www.avacs.org
[10] http://sensation-project.eu
[11] http://meals-project.eu

It benefited from scientific exchanges with EnergyBus e.V, with the CAN in Automation association, and with the emtas company. Ackowledgements are due to Dr. Wendelin Serwe for his advices on the best way of using the TGV tool, and to the anonymous reviewers for their comments about the present paper.

References

1. Vetter, M., Rohr, L., Ortiz, B., Schies, A., Schwunk, S., Wachtel, J.: Dezentrale netzgekoppelte PV-Batteriesysteme. In: VDI-Konferenz Elektrische Energiespeicher – Stationäre Anwendungen und Industriebatterien. (2011) 101–112
2. CAN in Automation International Users and Manufacturers Group e.V., EnergyBus e. V.: CiA 454 Work Draft Application profile for energy management systems – Document series 1 to 14, v. 1.0.6. (2012)
3. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., Lang, F., McKinty, C., Powazny, V., Serwe, W., Smeding, G.: Reference Manual of the LOTOS NT to LOTOS Translator (Version 5.8). Technical report, INRIA/VASY and INRIA/CONVECS (2013)
4. ISO/IEC: LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807 (1989)
5. ISO/IEC: Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001 (2001)
6. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. Software Tools for Technology Transfer (STTT) **15** (2013) 89–107
7. Jard, C., Jéron, T.: TGV: Theory, Principles, and Algorithms. Software Tools for Technology Transfer (STTT) **7** (2005) 297–315
8. Tretmans, J.: Model-based Testing with Labelled Transition Systems. In Hierons, R.M., Bowen, J.P., Harman, M., eds.: Formal Methods and Testing. Springer-Verlag (2008) 1–38
9. CAN in Automation International Users and Manufacturers Group e.V.: CiA 301 CANopen Application Layer and Communication Profile, v. 4.2.0. (2011)
10. CAN in Automation International Users and Manufacturers Group e.V.: CiA 305 Layer setting services (LSS) and protocols, v. 3.0.0. (2013)
11. CAN in Automation International Users and Manufacturers Group e.V., EnergyBus e. V.: CiA 454 Work Draft Application profile for energy management systems – Part 3: PDO communication, v. 1.0.2. (2012)
12. Graf-Brill, A.: Model-based Testing Approaches for the EnergyBus. Reports of SFB/TR 14 AVACS 96, SFB/TR 14 AVACS (2014) ISSN: 1860–9821, http://www.avacs.org.
13. Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A., eds.: Model-Based Testing of Reactive Systems – Advanced Lectures. Volume 3472 of Lecture Notes in Computer Science., Springer (2005)
14. van der Bijl, M., Rensink, A., Tretmans, J.: Compositional Testing with ioco. In Petrenko, A., Ulrich, A., eds.: FATES. Volume 2931 of Lecture Notes in Computer Science., Springer (2003) 86–100
15. Garavel, H.: OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In Steffen, B., ed.: Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal. Volume 1384 of Lecture Notes in Computer Science., Springer (1998) 68–84

16. Prenninger, W., Pretschner, A.: Abstractions for Model-Based Testing – Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004). Electronic Notes in Theoretical Computer Science **116** (2005) 59–71
17. van der Bijl, H.M., Rensink, A., Tretmans, G.J.: Atomic Action Refinement in Model Based Testing. Technical Report TR-CTIT-07-64, Centre for Telematics and Information Technology University of Twente, Enschede (2007)
18. Garavel, H., Viho, C., Zendri, M.: System Design of a CC-NUMA Multiprocessor Architecture Using Formal Specification, Model Checking, Co-simulation, and Test Generation. Software Tools for Technology Transfer (STTT) **3** (2001) 314–331
19. Tretmans, J., Brinksma, E.: TorX: Automated Model Based Testing – Côte de Resyste (2003)
20. Hessel, A., Larsen, K., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing Real-Time Systems Using UPPAAL. In Hierons, R., Bowen, J., Harman, M., eds.: Formal Methods and Testing. Volume 4949 of Lecture Notes in Computer Science. Springer (2008) 77–117
21. Bringmann, E., Krämer, A.: Model-Based Testing of Automotive Systems. In: ICST, IEEE Computer Society (2008) 485–493
22. Gerke, M., Ehlers, R., Finkbeiner, B., Peter, H.J.: Model Checking the FlexRay Physical Layer Protocol. In Kowalewski, S., Roveri, M., eds.: Formal Methods for Industrial Critical Systems. Volume 6371 of Lecture Notes in Computer Science. Springer (2010) 132–147
23. Rushby, J.: An Overview of Formal Verification for the Time-Triggered Architecture. In Damm, W., Olderog, E.R., eds.: Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'02), Oldenburg, Germany. Volume 2469 of Lecture Notes in Computer Science., Springer (2002) 83–106
24. Milbredt, P., Vermeulen, B., Tabanoglu, G., Lukasiewycz, M.: Switched FlexRay: Increasing the Effective Bandwidth and Safety of FlexRay Networks. In: Emerging Technologies and Factory Automation (ETFA), IEEE (2010) 1–8
25. Krause, J., Hintze, E., Magnus, S., Diedrich, C.: Model Based Specification, Verification, and Test Generation for a Safety Fieldbus Profile. In Ortmeier, F., Daniel, P., eds.: Computer Safety, Reliability, and Security. Volume 7612 of Lecture Notes in Computer Science. Springer (2012) 87–98
26. Goswami, D., Lukasiewycz, M., Kauer, M., Steinhorst, S., Masrur, A., Chakraborty, S., Ramesh, S.: Model-based Development and Verification of Control Software for Electric Vehicles. In: Proceedings of the 50th Annual Design Automation Conference (DAC'13), Austin, Texas, USA, ACM (2013) 96:1–96:9
27. Hartmanns, A., Hermanns, H.: Modelling and Decentralised Runtime Control of Self-stabilising Power Micro Grids. In Margaria, T., Steffen, B., eds.: ISoLA. Volume 7609 of Lecture Notes in Computer Science., Springer (2012) 420–439