A RIGOROUS APPROACH FOR SMART GRID SYSTEMS ENGINEERING USING CO-SIMULATION

Brett Bicknell[(a)], Karim Kanso[(b)], José Reis[(c)] , Neil Rampton[(d)], Daniel McLeod[(e)]

[(a,b,c)] Critical Software Technologies, UK
[(d,e)] Selex ES, UK

{[(a)]BBicknell, [(b)]KKanso, [(c)]JReis}@criticalsoftware.co.uk
{[(d)]Neil.Rampton, [(e)]Daniel.Mcleod}@selex-es.com

ABSTRACT
This paper reports on the progress of a case study exploring the application of simulation and formal methods to the development of a cyber-physical smart grid voltage control system. The control system is required to monitor voltage across the low-voltage network and adjust it accordingly to ensure it is within required bounds. Formal methods are used to ensure that the control system fulfils its requirements, and simulation is used to validate the system and its requirements. It is demonstrated that using both formal verification and validation within a single toolset provides both an increased level of assurance that the system is correct and reduced development costs due to early identification of errors. In essence the methodology described in this paper, when correctly applied, improves system level design at the initial phase of systems engineering.

Keywords: continuous models, formal methods, Modelica, Event-B

1. INTRODUCTION

The modern power distribution network requires more intricate control methods compared to the traditional top-down approach, due to the increased uptake of micro-generation and the requirement to reduce energy waste alongside growing demand. Traditional SCADA control methods using operator driven controls are not likely to be practical, and hence automation must be applied in the distribution network. Providing assurance on these automatic closed-loop control techniques is extremely challenging, due to the overwhelming number of scenarios and potential inputs that have to be considered. Empirical testing may be impracticable due to the durations required to achieve suitable results, and even then it provides no guarantee that a representative range of conditions have been uncovered. Solutions that are designed to react dynamically to such a wide array of inputs are also difficult to test on anything but a full deployment on a real network.

1.1. ADVANCE

Through the FP7 ADVANCE project (Advanced Design and Verification Environment for Cyber-physical System Engineering) Critical Software Technologies and Selex ES are investigating an alternative approach to better support the verification and validation of cyber-physical systems (Edmunds, Colley, and Butler 2012; Colley and Butler 2014). Cyber-physical systems are characterized as computing systems that control physical systems.

Within the ADVANCE toolset, discrete models of the control system and supporting elements are developed and verified, and then co-simulated with continuous models of the environment to validate the requirements. The advantage is twofold; first, the control system is verified using mathematical proof, to ensure the logic is sound and the requirements of the system are complete and consistent. The properties that the control system is expected to uphold – for example, maintaining voltage levels within given thresholds – are specified formally as invariants. Secondly, the control system is validated through simulation against realistic environmental inputs, during which any violation of the formal invariants is highlighted. To support the simulation activities, test cases are automatically generated which are used to check that suitable test coverage has been achieved. In all, this provides a higher level of assurance and confidence on the system before it is deployed or even physically tested. It has the potential to reduce engineering costs by identifying issues early in the system's lifecycle.

1.1.1. Discrete Models

The discrete models are specified using the Event-B modelling language (Abrial 2010). Event-B is developed over set theory, and is used to define abstract state machines consisting of events (transitions) and variables, where the events can change the state of the variables. This means that once a system is modelled in Event-B, it can be analysed formally, and unambiguously.

Within the ADVANCE project, the Rodin toolset is being actively developed. Rodin is an Event-B plugin for the Eclipse framework that allows for the development and analysis of Event-B models (Abrial et al. 2010). Moreover, Rodin provides the capability for formal proofs to be carried out within classical first order logic, which allows for the developer to mathematically determine whether certain properties of the system – i.e. invariants – hold under all circumstances.

It is also possible to explore the validity of the invariants through simulation and model-checking of Event-B models within Rodin. This is advantageous as it allows for not only determining whether the system violates a property, but also how it violated the property by providing an execution trace of the system.

1.1.2. Continuous Models

The continuous models are specified using Modelica, an open, object-oriented, multi-domain, modelling language (Fritzson 2010). Conceptually, Modelica is similar to the Simscape package of Simulink, in that it allows for physical systems to be specified by connecting various blocks together. For example, these blocks could represent a resistive electrical load, or rotational mechanics. Importantly, each block contains variables (discrete and continuous) and their defining equations, which are either simple equalities or differential equations. Modelica was chosen due to its open nature; however it is possible to use other tools to create the continuous models.

Using a Modelica simulation tool, such as OpenModelica, it is possible to simulate the physical model to determine how it behaves. The simulated model is inspected to investigate its behaviour over time, typically by plotting the time-evolution of different variables in the model. This provides feedback as to the correctness of the physical model; i.e. it is established whether the model exhibits the correct behaviour before co-simulation with the discrete models.

In this work, it is required that the Modelica models are converted into a binary format (that includes the differential equation solver), so that it is possible for Rodin to import and use them.

1.2. Overview

The paper is structured as follows: Section 2 introduces the case study around the low-voltage network control, Section 3 overviews related works, Section 4 discusses the modelling techniques required for the case study, and Section 5 gives an overview of the simulations undertaken and the results obtained so far. Section 6 provides concluding remarks.

2. CASE STUDY OVERVIEW

One of the case studies used to evaluate the ADVANCE methodology during the project concerns the development of an algorithm which provides automated control of the power distribution in a low-voltage network. The high level architecture of the case study is depicted in Figure 1.

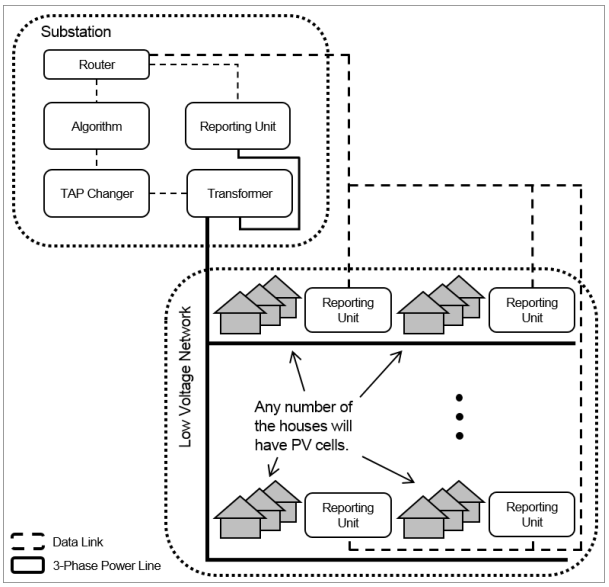

Figure 1: Case Study Architecture

A transformer with an On-Load Tap Changer (OLTC) is installed at a secondary substation, along with reporting units which are installed approximately half way and at the end of each feeder. The OLTC is capable of changing the transformer ratio through a number of discrete steps, providing stepped regulation of the output voltage. Reporting units measure the L-N (Live-to-Neutral) voltage for each phase of the feeder they are connected to and report their measurements to the substation.

The goal is for the algorithm to control the voltage on the low-voltage network efficiently, and ensure it remains within statutory limits. This is achieved by giving the algorithm the capability to set the target voltage setting for the OLTC, which in turn will select the most appropriate tap position to fulfil the target. The algorithm determines the target voltage for the OLTC by monitoring the voltage values provided by the reporting units.

Historically, there has not been a substantial issue with maintaining voltage levels across the network; transformer settings were determined during substation installation and rarely changed. However, with the increasing adoption of consumer micro-generation, typically in the form of photovoltaic cells, energy flow is now exhibited both towards and away from the substation. Additionally, increasing use of low carbon technologies, including electric vehicles and heat pumps, are increasing the demand on the network. The effect is a more dynamically changing voltage on the network, which increases the risk of the voltage in parts of the network going outside of the statutory limits. As the penetration of low carbon technologies increases over the coming decade, the need for more sophisticated

voltage management solutions will become ever more prevalent.

The algorithm considered in the case study is structured as described in Figure 2, where the busbar variable represents the voltage on the low-voltage side of the transformer. The intention is that the algorithm periodically reads the inputs, using reports which are sent by the reporting units monitoring different points of the network, and then every *k* time steps executes a number of rules that calculates the new target voltage for the OLTC.

```
Initialise()
do {
  ReadInputs()
  if (time mod k == 0) {
    ComputeMinMaxValues()
    if (max > A) {
      if (min < B) {
        target := busbar - X
      } else { if … }
    } else { if … }
    OutputTarget()
  }
 } while(sleep(1minute))
```

Figure 2: Algorithm Pseudo-Code

The properties of the system that need to be checked – which are specified in the models as Event-B invariants – are derived from the stakeholder requirement on the system to keep the voltage within required bounds at all points on the network. The challenge is in verifying the behaviour of the algorithm against these invariants, taking into account realistic power throughput as well as the effect of late or missing reports from the reporting units, and electrical faults on the network.

It is clear that scenarios such as this are archetypal of modern cyber-physical systems; where there are numerous computing systems all cooperating over potentially unreliable communications channels to achieve a common goal. Further, the unreliable nature of the communications mechanism means that it is challenging for formal methods alone to verify this system – typically a formal approach would assume that the communications mechanisms are reliable. In fact, this is where simulation plays a critical role to further assist in verifying the system.

In the following sections, it is considered what the discrete and continuous models in this scenario are, and how they are first verified formally and then simulated together to verify the system as a whole. But first, related works are considered.

3. RELATED WORK

It is well known that formal methods have been successfully applied to large industrial software systems, a recent survey of these has been compiled by Jim Woodcock (Woodcock et al. 2009). In an attempt to allow for formal methods to be applied to systems that are dependent on the physical wold (i.e. cyber-physical systems), formal hybrid methods were developed (Henzinger 2000). These are methods that combine discrete and continuous constructs. However, due to the complexity of these models and the resulting infinite nature, their application was mainly limited to academia with the notable exception of KeYmaera (Platzer and Quesel 2008).

In parallel, a simulation methodology evolved for developing systems that are tightly coupled with the physical world. Coming out of these simulation based methods was the need for combining heterogeneous models and reusing models (potentially between different tools). The solution to these needs became known as co-simulation. Ptolemy was one of the first tools to use co-simulated heterogeneous models, connected in an arbitrarily hierarchical fashion (Buck 1994; Chang, Kalavade, and Lee 1996). In recent years, two non-proprietary standards for co-simulation have emerged: FMI and DESTECS (Blochwitz et al. 2011; Pierce et al. 2012). These standards necessitate that the model to be simulated contains all the solvers that it requires, so it can be treated as a black box, and thus is portable between tools.

It was inevitable that co-simulation would be used as a mechanism to combine formal discrete models and continuous models (Živojnovic and Heinrich 1996; Fitzgerald et al. 2010). One directly relevant integration of these two approaches was by George Hackenberg, where formal modelling techniques were applied to the smart grid domain (Hackenberg et al. 2012). A bespoke environment based on the FOCUS approach (Broy and Stølen 2001) was used. This allowed for the creation of discrete models representing the control systems of the appliances in a number of houses. Models of the low and medium voltage power distribution network were also created, and simulated with the discrete models. The fundamental approach is similar to that applied in this work, however, it demonstrated a proof of concept and did not apply the tool to an industrial problem. In contrast, this work is assisting in the development of an industrial system.

Vitaly Savicks and Jens Bendisposto developed the plugin to the Rodin toolset that has allowed for the co-simulation of Event-B and Modelica models described in this paper (Savicks et al. 2013).

4. MODELLING

Elements of the system were modelled using both the Event-B and Modelica modelling languages. Event-B was used to model the algorithm, the reporting units, and an abstract communication network between these components. This created an encompassing discrete model of the cyber portion of the system. The physical models were modelled using Modelica, and consist of the low-voltage power network – including domestic customer demand models and inputs from photovoltaic sources – and the OLTC, complemented with a stochastic model specifying the occurrence of communication outages. The interaction between these

models is depicted in Figure 3, which also forms the basis of the simulation setup described in Section 5.

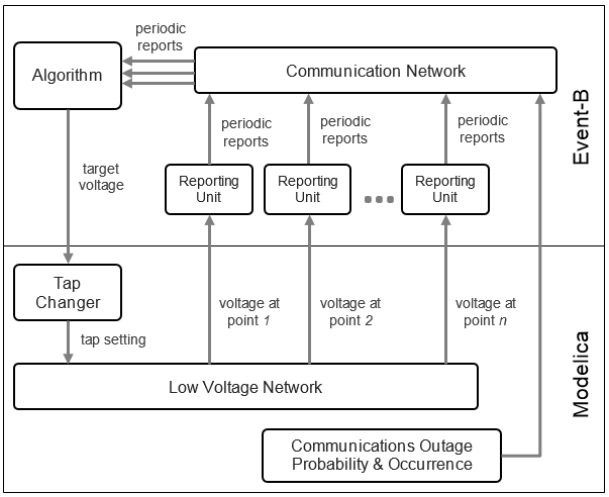

Figure 3: Modelling Strategy

From Figure 3, it can be observed that the discrete Event-B models represent the elements to be tested, whereas the continuous models represent the environment against which the system must be verified.

4.1. Event-B Models

Different strategies were required for modelling the Event-B models, each of which are detailed in this section.

4.1.1. Control Algorithm

The control algorithm was modelled using a state machine plugin for the Rodin framework, which allowed for a UML-style hierarchical approach to modelling the algorithm (Snook and Butler 2006). First, the high level state was considered, in which the algorithm can enter or leave a 'safe' mode. This safe mode is entered if the information the algorithm has about the network is not sufficiently up-to-date to make a decision; for instance, in the case that communication from a large number of reporting units is lost. A simplified example of a state machine representing this top-level view, generated in Rodin, is shown in Figure 4.

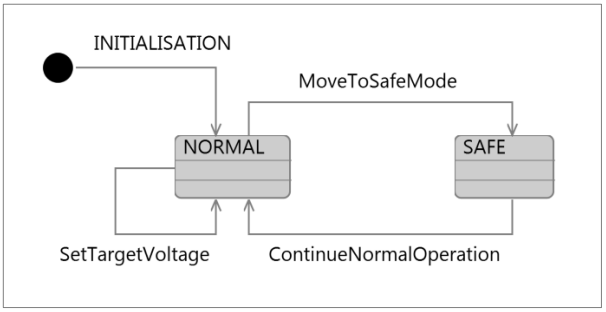

Figure 4: Event-B State Machine

These state machines can be translated to Event-B within Rodin, and further detail added to the transitions and states. A subset of the Event-B code corresponding to the state machine in Figure 4 is represented below in Figure 5.

```
variables
    state target comms_status

invariants
    @inv1 card(comms_status ▷ {FAILED}) ≥ 10
                                ⇒ state = SAFE

events
    event SetTargetVoltage
      any new_target
      where
        @grd1 state = NORMAL
      then
        @act1 target ≔ new_target
    end

    event ContinueNormalOperation
      where
        @grd1 state = SAFE
        @grd2 card(comms_status ▷ {FAILED}) < 10
      then
        @act1 state ≔ NORMAL
    end
```

Figure 5: Event-B code

The transitions in Figure 4 are translated into events, each of which has a number of guards and actions. The actions update the variables in the model, and the guards dictate when the events can occur. For example, the guard *@grd1* in the event *SetTargetVoltage* in Figure 5 ensures that the target voltage can only be changed when the algorithm is in normal mode. The action *@act1* in the same event assigns a new value to the variable that represents the target voltage.

Invariants are specified in each Event-B model, which represent the conditions, or requirements, that the model is expected to uphold. Proof obligations are generated to unambiguously determine whether each event maintains the invariants. For example, the invariant *@inv1* in Figure 5 represents the requirement that the algorithm has to enter the safe mode if the number of failed communication links exceeds a particular threshold. In order to prove that the event *ContinueNormalOperation* maintains this invariant, a guard has to be added to the event – *@grd2* – which only allows the model to exit the safe mode if the communication falls back below this threshold.

Further detail to the algorithm model was added through a number of additional modelling levels (refinements). This approach of defining a chain of models, each adding more detail, is archetypal of modelling in Event-B. It allows for the key elements of the system to be captured, whilst omitting any details which are not relevant to the properties being tested. An

example of a refinement of the *ContinueNormalOperation* event in Figure 5 is shown in Figure 6. Additional events, guards, actions, variables and invariants can be added at each refinement step.

```
event ContinueNormalOperation
  where
    @grd1 state = SAFE
    @grd3 card(mid_status ▷ {FAILED}) < 4
    @grd4 card(end_status ▷ {FAILED}) ≤ 6
    @grd5 timeout_exceeded = TRUE
  then
    @act1 state ≔ NORMAL
    @act2 counter ≔ 0
end
```

Figure 6: Refined Event

Further guards and actions have been added to the event in Figure 6 as part of the refinement. In addition, guard *@grd2* of the abstract event in Figure 5 has been replaced by the more concrete guards *@grd3* and *@grd4*, which represent the communication status of the mid and end points of the network separately. Further proof obligations are generated in the model in order to determine the consistency between the refinement levels. These proof obligations (once discharged) ensure that it is not possible to violate the invariants of more abstract levels of the model. This allows for the models to be produced in an iterative fashion alongside design or requirements generation; additional features or requirements can be added to the model as a refinement, and insurance can be generated that they are consistent with the requirements already in place.

4.1.2. Communications Network

The communications network was modelled directly in the Event-B language. Different network topologies were considered for comparison and evaluation, including a centralised and mesh network. Modelling the communications network followed a similar top-down approach, in which the first model is a generic communications model of abstract send and receive events, to which details of the particular network topology were added in subsequent refinements. Moreover, adding details into subsequent models in this manner has allowed for model reuse. Different communication networks – which use different protocols – can be (and have been) branched off the main development.

4.1.3. Reporting Units

The reporting units used a slightly different strategy, namely, decomposition. This followed an approach where the wider system (including data centres) was first abstractly formalised, then decomposed into sub-systems. Only one of these sub-systems was further developed into a model of the reporting unit. This approach of first defining the system, and then extracting the portion of the system that is to be realised ensured that the reporting units fulfilled the requirements of the wider system. The decomposition is facilitated by a Rodin plugin, developed during the ADVANCE project.

4.1.4. Formal Proof

As mentioned in the previous sections, formal proof was used to verify the correctness of the Event-B models. This not only ensured that properties internal to each element (algorithm, reporting units and communication network) were preserved correctly, but also highlighted several issues with the system – some of which were previously unknown – as a result. These issues were highlighted due to the inability to discharge the relevant proofs.

4.2. Modelica Models

The three Modelica models – the low voltage network, OLTC and specification of communication outages – were developed using different techniques: equations, state machines, and stochastic specification.

The low voltage network model encapsulates the topology of the power network – i.e. 11kV power source, transformer, feeders, houses and voltmeters – along with the standard electrical equations of power networks. A block representation of the top level model of the low voltage network is depicted in Figure 7. In addition to Figure 7, the models also contain data that represents consumption and production of power for each consumer. This data is generated by incorporating the probabilistic outputs of the CREST project into the model (Richardson 2011). The CREST models are very detailed and consider the number of occupants, when and which domestic appliances are used, when lights are used, and PV contributions.

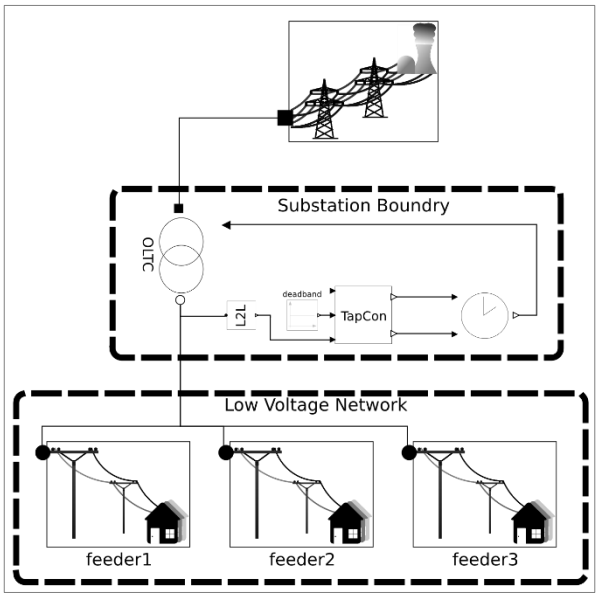

Figure 7: Low Voltage Top Level Model

The OLTC model (represented by the TapCon block in Figure 7 and elaborated in Figure 8) monitors the voltage at the transformer and changes the tap position when the voltage deviates from the target by more than a specified amount, mimicking the automated behaviour of the actual OLTC. Although the OLTC contains both discrete and continuous behaviour, it was modelled in Modelica as it is an off-the-shelf device representing part of the environment for the Event-B models. The OLTC was modelled using the Modelica state machine library, and is depicted in Figure 8, where the boxes represent states and the black bars represent transitions.

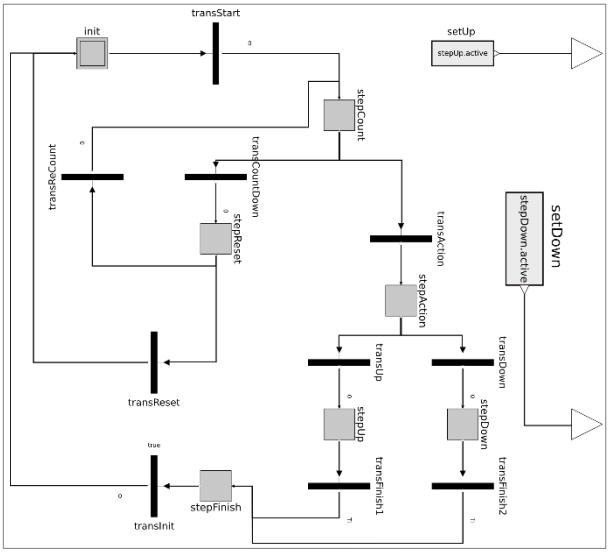

Figure 8: OLTC State Machine

The communication outage model is a stochastic model in Modelica. Based on industrial statistics it determines at what time, and for how long, a given communication link fails. The implementation of the stochastic models within Modelica is canonical, and represented by a sequence of probabilities (and associated actions) that partition the interval [0,1], then at discrete time steps, using the Peter Fritzson pseudo random number generator, generate a number in [0,1] and select the corresponding action from the sequence. In this case the action determines how long a communication link fails for.

Using industrial stochastic based simulation of the communication link failures provides a fast and efficient means to analyse the system for robustness under realistic operating conditions.

5. SIMULATION

Within the ADVANCE framework there are two types of simulation. The first, simulates the discrete models using the ProB model-checker. The second, simulates the discrete and continuous models together and is known as co-simulation.

5.1. Discrete Model Animation

To aid in verification of the discrete Event-B models, the model-checker tool ProB is used (Leuschel and Butler 2003). This allows for the models to be executed within the Rodin environment to explore their behaviour. This provides instant feedback to the model developers as to whether the models have the intended semantics. It is possible to use the model-checker to analyse the models and determine whether they deadlock or violate requirements. In either case, when one of these scenarios is identified the model-checker outputs an execution trace from the initial state of the model to the state that violates the condition. This allows for the model developer to quickly identify cases where the model does not behave as intended, fix the problems, and reanalyse. An example of this is shown in Figure 9 below. A violated condition (invariant) has been found and is highlighted in bold. The trace of events that lead to this violation is displayed in the *History* tab. The state of the variables in the model can also be examined at each step of the trace.

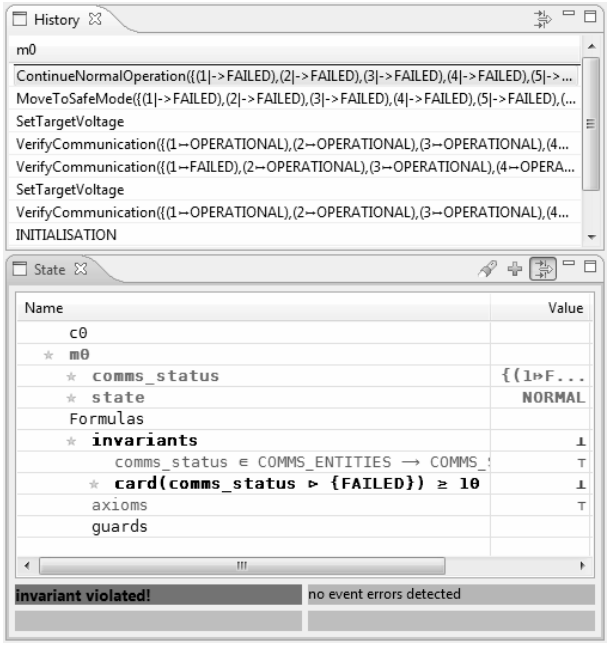

Figure 9: ProB Simulation

It is also possible to define graphical animations that depict the state of the system while it is being simulated in ProB. For example, in the case of the mesh communications network mentioned previously, an animation was developed that showed the nodes, the status of the communications links, and the volume of packets transiting each link. This is depicted in the example in Figure 10, where the boxes are nodes, the active links are represented by the solid lines, and the width of these lines represents the volume of data traffic on the link.

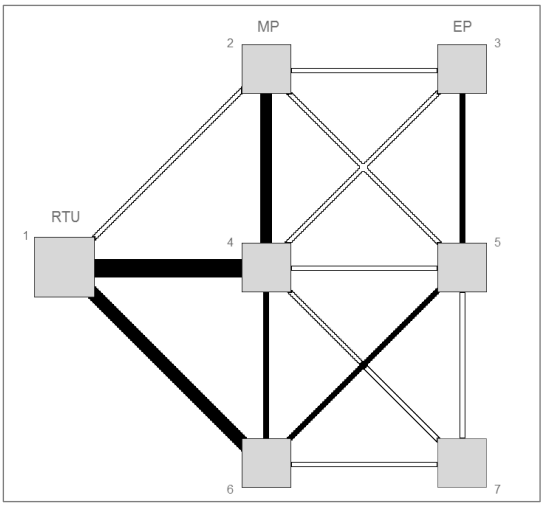

Figure 10: Mesh Network Animation

However, this type of discrete model simulation does not explore whether the models behave correctly when in situ, and the wider question of system validation is not considered – it is geared towards verifying that the models fulfil their requirements.

5.2. Co-Simulation

To validate that the system fulfils its intended purpose, the use of co-simulation is employed. The Event-B models and Modelica models are simulated in parallel, such that the outputs of one model become the inputs of another model. This is achieved through the tools developed within the ADVANCE project (Savicks 2013), and is fully integrated into the Rodin environment.

The architecture of the simulation framework is depicted in Figure 11. It is possible to integrate a number of Event-B and Modelica models into a single simulation. The configuration is used to define which inputs and outputs are linked together, and the size of the time steps of the models. Thus the simulation engine manages global temporal aspects of the simulation, it tells each model when to execute, and in the case of the Modelica models for how long. See Figure 12 for pseudo code depicting the functionality of the engine. The *performIO* step is responsible for copying values between input and output ports of the models; this is called for each model before any model starts execution, to ensure that the outputs are not overridden with new values before they are read by the other models. The current implementation of the engine does not allow for one model to interrupt the simulation, e.g. in response to an event. To mitigate this issue, the time steps of the models must be set sufficiently small enough to respond to outputs from other models.

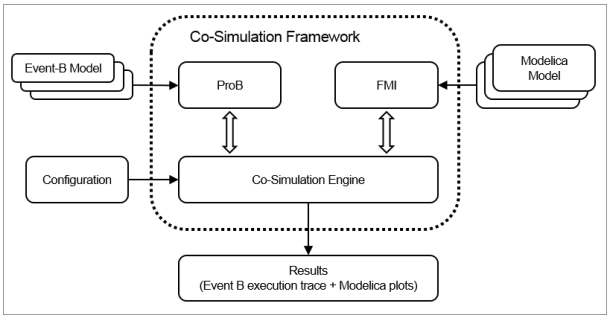

Figure 11: Conceptual View of Co-Simulation

```
int time := 0
while (time < endTime) {
  List modelsToEvaluate := empty
  int nextTime := endTime
  for m in models {
    if (time >= m.executeTime) {
      modelsToEvaluate.append(m)
      m.performIO()
    }
  }
  for m in modelsToEvaluate {
    m.execute(time,m.stepSize)
    m.executeTime += m.stepSize
    nextTime := min(nextTime,m.executeTime)
  }
  time := nextTime
}
```

Figure 12: Simulation Algorithm Pseudo Code

The configuration of the co-simulation within this case study is defined in Figure 3, where the arrows depict the linking of an output port on one model to the input port on another model. Additionally, timing configuration data is required. This details how each model perceives the progress of time with respect to the global time. For instance, the reporting units have a much higher clock cycle time than the algorithm cycle time, and the communication network needs to operate with smaller intervals to ensure packets are delivered at a realistic pace. The Modelica models are less dependent upon time, but still need to know how much time has passed and the length of the computation step required. Here, the low voltage network model operates at the same time intervals as the reporting units, to ensure that the information detected by the sensors is current.

The discrete models are simulated using the same technique as described previously, namely, with the ProB model-checker. Whereas, the continuous nature of Modelica models require the use of differential equation solvers. These solvers are embedded into the models when the model is compiled into a functional mock-up unit (FMU). An FMU is a binary representation of a model, typically produced by a Modelica toolset (Blochwitz et al. 2011).

Within Rodin, once the models have been configured, the simulation begins, which results in two

artefacts: time series for chosen variables and execution traces of the discrete models.

Within the domain of simulation, time series are common place, but taking advantage of the underlying framework of Rodin it is also possible to drill down into the execution traces of the discrete components. Thus, it is possible to understand exactly why a model makes a decision. The inputs to the model are plotted, the internal state of the model is examinable and the commands it executed are recorded. This provides a high-fidelity view of the simulation, which other toolsets often lack.

5.2.1. Results

The initial simulations pinpointed serious issues with the discrete model of the algorithm entering and leaving the designated safe mode at the wrong times. This is interesting as the models were formally developed; it in fact underlines the issue that the requirements did not fully define the safe mode behaviour. This was identified by noticing that the target was not set correctly in the simulations, then further investigation into the execution traces showed that the algorithm was entering safe mode unnecessarily. After modifying the conditions for entering safe mode, this issue was fixed.

Subsequent simulations indicated under which conditions the solution could maintain the voltage bounds. Under ideal circumstances the algorithm was able to maintain the voltage; however, when the feeders were unbalanced the voltage on some feeders could not be maintained within the UK regulated +10%/-6% bounds. This is depicted in Figure 13, where the thick line denotes the lower bound at 216. In this example the upper bound, at 253, was not breached.

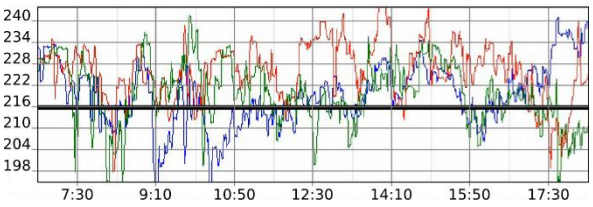

Figure 13: Feeder Violations (V)

Further observations relating to the general behaviour of the system were also raised. The system is heavily dependent on the medium voltage input. A simulated medium voltage variation (from a nominal 6.7 L-N KV or 11 L-L KV) is depicted in Figure 14, and is seen to be mirrored in the low voltage transformer output in Figure 15. Figure 15 also shows the relationship between the target set by the algorithm (dotted blue line) and the busbar voltage (on the low-voltage side of the transformer), which is as expected; notably not every target change corresponds to a tap change. The correlating changes in tap position for this example are shown in Figure 16; each discrete increment changes the busbar L-L (Line-to-Line) voltage by 8V.

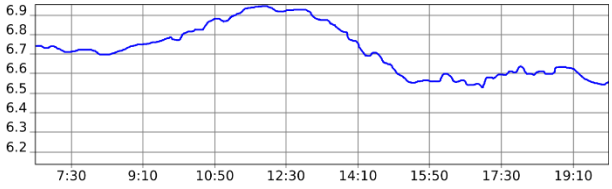

Figure 14: MV Input (KV)

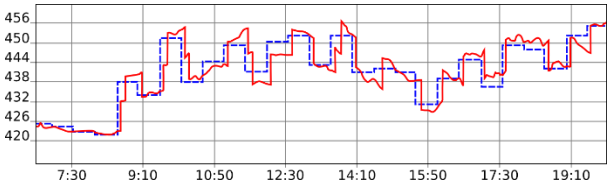

Figure 15: Busbar and Target (L-L Voltage (V))

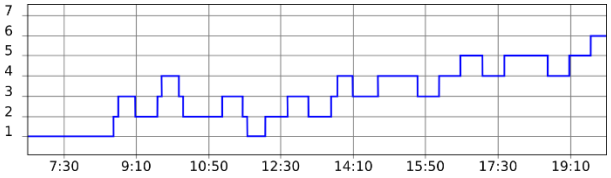

Figure 16: Tap Position

6. CONCLUSION

In this paper, the low-voltage smart energy case study of the ADVANCE project has been described and the initial results presented. The paper also overviewed the ADVANCE methodologies, with special emphasis on cyber-physical systems engineering.

The advantages of the ADVANCE framework are twofold:

- Formal development and verification of control systems, and
- Validation of the developed system against plant models.

Further, a single development environment is used for the formal development and analysis, with the freedom to integrate with any FMU plant model. In this paper, all plant models were developed using Modelica.

Changes are made to the formal models if issues are found through the verification or validation, and retested against the environment until a satisfactory solution is found.

This allows for the system to be formally analysed, simulated and improved at an early phase of the system life cycle, hence reducing time to market of complex systems.

ACKNOWLEDGMENTS

We would like to thank members of the ADVANCE consortium for their continued support throughout this work, especially the University of Southampton and Heinrich Heine University Düsseldorf for their continued support with the modelling and co-simulation.

We would also like to thank the anonymous reviewers of this paper for their time and helpful comments.

REFERENCES

Abrial, J.-R., 2010. *Modeling in Event-B: system and software engineering*. UK:Cambridge University Press.

Abrial, J.-R., et al., 2010. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*. volume 12:447-466.

Blochwitz, T., et al., 2011. The functional mockup interface for tool independent exchange of simulation models. *Proceedings of 8th Modelica Conference*, pp. 105-114. March 20-22, Dresden, Germany.

Broy, M. and Stølen, K., 2001. *Specification and development of interactive systems: focus on streams, interfaces and refinement*. USA:Springer-Verlag New York.

Buck, J., Lee, E. and Messerschmitt, D., 1994. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *Int. Journal of Computer Simulation, special issue on "Simulation Software Development"* volume 4:155-182.

Chang, W.-T., Kalavade, A. and Lee, E., 1996. Effective Heterogenous Design and Co-Simulation. *NATO ASI Series: Hardware/Software Co-Design* volume 310:187-212.

Colley, J. and Butler, M., 2014. *Advanced Design and Verification Environment for Cyber-physical System Engineering*. Web. Available from: http://www.advance-ict.eu/ [May 2014].

Edmunds, A., Colley, J. and Butler, M., 2012. Building on the DEPLOY legacy: code generation and simulation. *DS-Event-B-2012: Workshop on the experience of and advances in developing dependable systems in Event-B.*

Fitzgerald, J., Larsen, P., Pierce, K., Verhoef, M. and Wolff, S., 2010. Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems. *Lecture Notes in Computer Science*. volume 6396:12-26.

Fritzson, P., 2010. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons.

Hackenberg, G., Irlbeck, M., Koutsoumpas, V. and Bytschkow, D., 2012. Applying formal software engineering techniques to smart grids. *Proceedings of 1st International Workshop on Software Engineering for the Smart Grid (SE4SG)*, pp. 50-56. June, Zurich, Switzerland.

Henzinger, T., 2000. The Theory of Hybrid Automata. *NATO ASI Series: Verification of Digital and Hybrid Systems* volume 170:265-292.

Leuschel, M. and Butler, M., 2003. ProB: A Model Checker for B. *Lecture Notes in Computer Science*. volume 2805:855-874.

Pierce, K., et al., 2012. Collaborative Modelling and Co-simulation with DESTECS: A Pilot Study. *Proceedings of 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 280-285. June, Toulouse, France.

Platzer, A. and Quesel, J.-D., 2008. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). *Lecture Notes in Computer Science: Automated Reasoning* volume 5195:171-178.

Richardson, I., 2011. *Integrated high-resolution modelling of domestic electricity demand and low voltage electricity distribution networks*. PhD Thesis. Loughborough University.

Savicks, V., Butler, M., Bendisposto, J., and Colley, J., 2013. Co-simulation of Event-B and Continuous Models in Rodin. *Proceedings of 4th Rodin User and Developer Workshop*. June, Turku, Finland.

Snook, C., Butler, M., 2006. UML-B: Formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology* volume 15 issue 1:92-122.

Woodcock, J., et al., 2009. Formal Methods: Practice and Experience. *ACM Computing Surveys* volume 41 issue 4:1-40.

Živojnovic, V. and Heinrich, M., 1996. Compiled HW/SW co-simulation. *Proceedings of the 33rd annual Design Automation Conference*, pp. 690-695. June, Las Vegas, USA.

AUTHORS BIOGRAPHY

Brett Bicknell holds a BSc Physics degree and has played a key role in a number of formal methods initiatives, including the FP7 ADVANCE project. His software engineering experience encompasses varying levels of criticality, including embedded systems and data solutions.

Karim Kanso has worked within the field of formal methods and software engineering for many years, on various projects in the domains of transportation and aerospace, and received a PhD in theoretical computer science.

José Reis is a Principal Consultant Engineer at Critical Software Technologies. He plays a key role in the development of model-driven engineering and formal methods within Critical Software Technologies. He has been leading the team working on the ADVANCE FP7 project and prior to that Mr. Reis was involved with DEPLOY FP7. In parallel with this Mr. Reis has been working with various prime contractors, such as BAE Submarines, EADS Astrium and AgustaWestland, with a focus on requirements engineering and verification and validation.

Neil Rampton is a professional electronics engineer who has worked for 25+ years within Selex ES. Neil has experience in a wide variety of senior engineering roles, including within engineering management, business/solutions development, programme management and change management. His project

involvement has included a wide variety of electronics and systems, including high performance thermal imaging cameras and Generic Vehicle Architectures for military vehicles. He is currently responsible for Capability and Product Strategy for Smart Energy (Smart Grid and Buildings Energy Management) and is looking at the development of Smart City & Transportation solutions.

Daniel McLeod holds a BSc Physics degree and has worked on a variety of imaging sensor systems in the Airborne, Land and Naval domains. His previous work includes algorithm development, system design, requirements analysis and verification and project management. He is currently working within the smart energy domain on a Low Voltage monitoring system and a system for automated control of low voltage transformers in response to customer demand and distributed generation.