Modelling and Analysing an Industry 4.0 Communication Protocol

Benjamin Aziz *Member, IEEE*

***Abstract*—The increasing complexity and criticality of industrial automation systems, embodied by the concept of Industry 4.0, which brings together concepts like Cyber-physical systems, the Internet-of-Things, Big Data and Artificial Intelligence, calls for more formality in specifying the technology standards underlying these systems. We define in this paper a formal model of an Industry 4.0 machine-2-machine communication protocol, Hermes, used in specifying electronic board transfers in an assembly line. Our analysis of the formal specification reveals that, despite the robustness of the protocol, many testing scenarios have been ignored in the protocol standard and in particular, scenarios that include simultaneous machine errors. Therefore, our work paves the way for a better informed testing strategy in Industry 4.0 systems that implement the protocol.**

***Index Terms*—Communication Protocols, Cyber-Physical Systems, Formal Analysis, Industry 4.0, IoT.**

I. Introduction

The new wave of digitisation in manufacturing, dubbed Industry 4.0, brings to the manufacturing industry the benefits of many of the recent technological paradigms including Cyber-Physical Systems, Internet-of-Things, Cloud computing and Artificial Intelligence. This has lead to smarter and more autonomous systems. Of these, the IPC-HERMES-9852 standard [1] (Hermes, henceforth) has emerged as a leading communication protocol in the electronics manufacturing industry, and as a replacement for the IPC-SMEMA-9851 Mechanical Equipment Interface Standard [2]. The Hermes protocol describes necessary steps in transporting a Printed Circuit Board (PCB) from one machine to another and a description of the scenarios that may occur in the event of errors detected at either the sending or the receiving machine. Hermes is maintained by the Institute for Printed Circuits (IPC) and it is hailed as a non-proprietary open protocol bringing the benefits of Industry 4.0 to the electronics community.

As with any other protocol, more clarity and understanding of the properties of the protocol and its operation can be gained by following formal specification and analysis techniques. In this paper, we specify the Hermes protocol formally using a version of the π-calculus language [3]. We also apply an abstract interpretation to analyse the protocol for agreement properties. These include agreement on the completeness, non-completeness and non-starting of the PCB transfer between the sending and receiving machines in the protocol.

We summarise our two major findings as follows:

B. Aziz is with the School of Computing, University of Portsmouth,

- Specifying the protocol formally has shown that a number of error-detection scenarios were missed by the standard document [1], as part of the protocol's behaviour. The document identified only 7 such scenarios, whereas we found that there were 24 scenarios that needed consideration. The significance of this finding is that it will lead in the future to more comprehensive testing suites applied to software implementations of the protocol by increasing the coverage of these tests from 7 cases to 24. This will further increase the robustness of those implementations as a result. This basically means that implementations of the protocol ought to be tested more than what the current standard recommends in [1].
- Related to the above finding, the standard document [1] does not consider simultaneous occurrences of errors – only that a single machine detects an error at any single point in time. This is indicated for each and every one of the 7 scenarios highlighted in the document. However, in reality, such errors may occur simultaneously, and therefore it is necessary also to consider the simultaneous triggering of the error-handling processes at both the sending and receiving machines. This, as we show later in more detail, could lead to these processes interacting with one another.

The rest of the paper is structured as follows. In Section II, we discuss related work in literature. In Section III, we give some background on the Hermes protocol and the formal language used in the modelling and analysis. In Section IV, we define formally the Hermes protocol for the case of normal operation and analyse the correctness of the protocol in relation to the PCB transfer completion property. In Section V, we extend this definition to be able to deal with errors, analysing for the cases of non-started and incomplete PCB transfers. Finally, in Section VI, we conclude and highlight future research.

II. Related Work

Formal specification and analysis techniques have been used extensively ever since their inception as a method for increasing the dependability of systems and deepening our understanding of the behaviour of critical systems. Industry 4.0, which brings together several paradigms including Cyber-Physical Systems, the Internet-of-Things (IoT), Cloud computing, Big Data and Artificial Intelligence among others, poses particular challenges due to the high dimensionality and complexity of the systems involved. Formal methods have much to contribute to the demystification of such complexity [4]. In fact, the important role that formal methods can play in

industry to enhance dependability and reliability of industrial systems was emphasised in works as early as [5], [6], [7], needless to say that this strong relationship emerged between the two worlds in the form of infamous methods like VDL [8], Z [9] and B [10]. Examples of early surveys highlighting cases where formal methods contributed to the enhancement of industrial systems include [11], [12], [13], [14], [15].

More recently, IoT protocols have had a direct relationship with industrial systems, both together sometimes referred by the term Industrial IoT (IIoT). Amongst the most widely modelled and formally verified IoT protocols is the MQTT protocol [16]. Various works, e.g. [17], [18], [19], [20], [21], have addressed issues with the reliability and security of the protocol using a variety of formal methods. Most notably, in [17], [18], it was discovered that the informal semantics of the protocol contained subtle ambiguities that undermined the protocol's correctness. Other IoT protocols that have benefited from the application of formal methods include CoAP [22], which has been verified in a number of works including [23], who used the Event Calculus [24] to generate a set of monitoring events for the run-time verification of CoAP systems. By contrast, [25] used model checking techniques to analyse the flow of messages among CoAP nodes.

Apart from this work, other works have also addressed the problem of formally modelling the properties of Industry 4.0 communication technologies. In [26], the authors studied some of the security properties of the Open Platform Communications (OPC) Unified Architecture (UA) [27] using the ProVerif static analysis tool [28]. One of the weaknesses of such tools, in addition to the complexity underlying their usage, is their reliance on the presence of explicit cryptography in the protocol in order to formalise security properties. As such, [29] used UML to semi-formally model UA.

In [30], the authors also used UML, particularly UML activities [31], as part of the Reactive Blocks [32], [33] model-driven approach to model control software in industrial automation. Similarly UML was combined with net conditions/event systems [34] to model and validate automation systems in [35], particularly plants' structures and dynamics. The formal specification part facilitated the checking of properties such as the lack of dangerous situations in the plant, robustness of the system in terms of the malfunctioning of sensors, avoidance of deadlocks and particularly of relevance to our work in this paper, the presence of ample checkpoints in any possible scenario of behaviour the system may follow.

Security has also been the subject of the application of formal verification techniques in the context of Industry 4.0 systems, particularly when it comes to the issue of the integration of various system components. In [36], for example, the Shibboleth federated identity and single-sign on protocol [37] was proposed as a solution to securely connect Fog clients and Fog nodes [38] in automation systems. High-level Petri nets [39] were then used to demonstrate the reliability and robustness of this connection. The various security issues that can arise in Industry 4.0 systems were highlighted in [40].

The approach adopted in this paper follows from a number of existing works [41], [17], [17], [18], which demonstrated the effectiveness of the method in analysing properties and detecting issues with different systems and protocols. The original theory underlying the abstract analysis framework was defined in [42], to which we refer the reader for further detail.

To conclude, existing literature highlights the wide context in which formal approaches can be utilised to improve the specification of modern industrial systems and clarify ambiguities in their designs in relation to a variety of properties, such as reliability and security. It is this context, and its importance, that has largely motivated the work presented in this paper.

III. Background

We give in this section some background on the specification of the Hermes protocol and the formal language used in modelling the protocol.

A. The Hermes Protocol

The normal operation of the Hermes protocol is illustrated in Figure 1, with alternative sequences of handshakes.

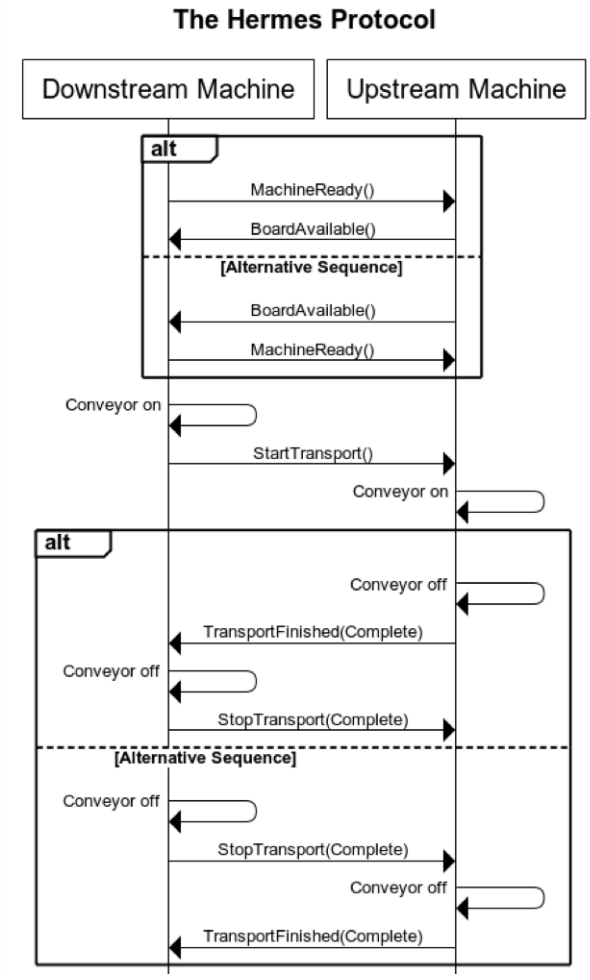

Fig. 1. The normal operation of Hermes [1] with alternative sequences.

An Upstream machine is a machine that sends a PCB whereas a Downstream machine is a machine that receives the PCB. Note that the protocol provides for two alternatives for two of its message blocks. The first is related to the exchange

of the first two messages, MachineReady() and BoardAvailable(), which simply allow each machine to indicate to the other that it is ready to commence the PCB transfer. After that, the Downstream machine starts the transfer by sending the StartTransport() signal. The second alternative sequence then commences and it is related to the final stage of the transfer, where either machine again can indicate before the other that it has completed the transfer. This is done using the pair of signals TransportFinished(Complete) and StopTransport(Complete) signals. All across the protocol timeline, either machine can switch on and switch off its conveyor belt by performing Conveyor on and Conveyor off internal actions, respectively, at the appropriate points in time. Further details of the sequence of messages and their meaning can be found in [1], which also describes a few error-detection scenarios.

It is worth noting that despite the fact that Hermes is, in its essence, an Industry 4.0 machine-2-machine protocol, it nonetheless provides for connectivity with the Internet, since both the upstream and downstream machines can offer a TCP server that can be contacted on port 1248 for the purpose of receiving configuration messages to configure the machines. This facilitates the deployment of a Hermes-based manufacturing line in an Industrial Internet-of-Things set-up.

B. The Process Algebra

The formal model of Hermes that we introduce later is based on a version of the π-calculus process algebra [3], which is a synchronous message-passing model. We give an overview of the syntax and semantics of this algebra.

C. Syntax and Structural Operational Semantics

We use the following syntax to define processes, $P, Q \ldots \in \mathcal{P}$, where names are denoted as $x, y \ldots \in \mathcal{N}$:

$$P, Q ::= \overline{x}\langle y\rangle.P \mid x(y).P \mid \tau.P \mid !P \mid (\nu x)P \mid (P \mid Q) \mid (P+Q) \mid \mathbf{0} \mid A \mid P[x=y]Q$$

The syntax corresponds to the synchronous π-calculus [3]. A process can perform any of the following actions: it can output a message over a channel, $\overline{x}\langle y\rangle.P$, it can input a message over a channel and use it to replace a parameter, $x(y).P$ and it can perform a silent internal action $\tau.P$. A process can also restrict the scope of a name $(\nu x)P$ and can be combined with another process in parallel, $(P \mid Q)$, or in a non-deterministic choice, $(P+Q)$. A process can also be replicated, $!P$, to generate as many copies of P as required or it can be passive and therefore do nothing, $\mathbf{0}$. We can also call a definition of a process by performing A where $A \stackrel{\text{def}}{=} P$. Finally, the special construct, $P[x=y]Q$, will evaluate the condition $x = y$ and if true, it will continue as P, otherwise as Q.

We omit the input parameter y in $x(y).P$ and write $x().P$ if no message is expected. Similarly, we omit the message y in $\overline{x}\langle y\rangle.P$ if no such message exists and simply write $\overline{x}\langle\rangle.P$. We call the set of free names of a process, $fn(P)$, such as x and y in $\overline{x}\langle y\rangle.P$, $x(u).P$ and $P[x=y]Q$, and the set of bound names, $bn(P)$, such as u and y in $x(u).P$ and $(\nu y)P$.

The structural operational semantics of the above version of the π-calculus is given in terms of the structural congruence, $\equiv$, and the operational transition, $\xrightarrow{\tau}$, relations as shown in Figure 2, where $fn(P)$ are the set of free names of P.

Rules of the $\equiv$ relation:

(1) $(\mathcal{P}/\equiv, |, \mathbf{0})$ is a commutative monoid
(2) $(\nu x)\mathbf{0} \equiv \mathbf{0}$
(3) $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
(4) $!P \equiv P \mid !P$
(5) $(\nu x)(P \mid Q) \equiv (P \mid (\nu x)Q)$ if $x \notin fn(P)$
(6) $P[x=y]Q = \begin{cases} P & \textit{if } x = y \\ Q & \textit{otherwise} \end{cases}$
(7) $A \equiv P$, where $A \stackrel{\text{def}}{=} P$

Rules of the $\xrightarrow{\tau}$ relation:

(8) $\overline{x}\langle y\rangle.P \mid x(z).Q \xrightarrow{\tau} P \mid Q[y/z]$
(9) $\overline{x}\langle\rangle.P \mid x().Q \xrightarrow{\tau} P \mid Q$
(10) $\tau.P \xrightarrow{\tau} P$
(11) $P \xrightarrow{\tau} Q \Rightarrow (\nu x)P \xrightarrow{\tau} (\nu x)Q$
(12) $P \xrightarrow{\tau} P' \Rightarrow P \mid Q \xrightarrow{\tau} P' \mid Q$
(13) $P \xrightarrow{\tau} P' \Rightarrow P + Q \xrightarrow{\tau} P'$
(14) $P \xrightarrow{\tau} P' \Rightarrow Q + P \xrightarrow{\tau} P'$

Fig. 2. The structural operational semantics of the π-calculus.

The significance of these relations is that they demonstrate how a process evolves. The definition of $\equiv$ expresses how the syntactic structure of the process can be altered without any operational evolution. On the other hand, the definition of $\xrightarrow{\tau}$ outlines how the process operates. The main rules here are rules (8) and (9), which define how communications take place resulting in the substitution $[y/z]$ when a message is passed (Rule (8)), and no substitution when no message is passed (Rule (9)). More detail on the semantics of the π-calculus can be found in works like [3], [43].

On the other hand, in [42], [44], we defined a non-standard name-substitution semantics for the π-calculus, which when abstracted using an approximation function, was capable of yielding an abstract environment:

$$\phi : \mathcal{N}^\sharp \to \wp(\mathcal{N}^\sharp) \in D^\sharp_\perp$$

where $\mathcal{N}^\sharp$ represents the set of abstract names. Unlike $\mathcal{N}$, $\mathcal{N}^\sharp$ is finite and as a result any element of the powerset of $\mathcal{N}^\sharp$, i.e. $\wp(\mathcal{N}^\sharp)$, is also finite. The resulting semantic domain, $D^\sharp_\perp$, guarantees termination for an abstract interpretation computed over it (with applications e.g. such as in [45]). The bottom element of $D^\sharp_\perp$, $\perp = \phi_0$, is the empty environment where $\forall x \in \mathcal{N}^\sharp : \phi_0(x) = \{\}$. $\mathcal{N}^\sharp$ and consequently $D^\sharp_\perp$ can be constructed using some approximation function that keeps the size of $\mathcal{N}^\sharp$ finite by limiting the number of copies of input parameter names and freshly created names.

The static analysis of a process, P, can then be obtained through the application of a suitable abstract interpretation function, $\mathcal{A}[\![P]\!]\phi_0 \in D^\sharp_\perp$. When analysing the specification of a process, we use the resulting ϕ environment to formalise

the definitions of the properties we are interested in. As a simple example, consider the following process:

$$P \stackrel{\text{def}}{=} (!\ x(y).\mathbf{0}) \ \mid \ (!\ (\nu z)\overline{x}\langle z\rangle.\mathbf{0})$$

were we could apply an approximation that limits the number of copies of the input parameter y and the freshly generated message z, to a maximum of *two* such copies. Then we would obtain the following abstract environment, $\mathcal{A}[\![P]\!]\phi_0 = \phi_{\textit{example}}$, where:

$$\phi_{\textit{example}}[y_1 \mapsto \{z_1, z_2\}, y_2 \mapsto \{z_1, z_2\}]$$

Defining any property amounts to the definition of a suitable predicate, $\digamma : D^{\sharp}_{\perp} \to \mathbb{B}$, on the resulting abstract environment. For example, we could choose to say that y will be instantiated with its corresponding copy of the z message:

$$\digamma(\phi_{\textit{example}}) = z_1 \in \phi_{\textit{example}}(y_1) \ \wedge \ z_2 \in \phi_{\textit{example}}(y_2)$$

which is true. We keep sub-environments resulting from choices of process executions explicitly separate for clarity. For example, consider the following process:

$$Q \stackrel{\text{def}}{=} x(y).\mathbf{0} \mid (\overline{x}\langle z\rangle.\mathbf{0} + \overline{x}\langle t\rangle.\mathbf{0})$$

When analysed, $\mathcal{A}[\![Q]\!]\phi_0 = \phi_{\textit{example2}}$, this would result in the following two sub-environments:

$$\phi_{\textit{example2}} = \phi'_{\textit{example2}}[y_1 \mapsto \{z\}] \ \cup \ \phi''_{\textit{example2}}[y_1 \mapsto \{t\}]$$

which we prefer to keep separately, instead of writing:

$$\phi_{\textit{example2}}[y_1 \mapsto \{z, t\}]$$

IV. A Formal Model of the Hermes Protocol

The formal model for the Hermes protocol reflects the two machines specified in the Hermes protocol [1]: the *upstream* and the *downstream* machines. These are formally defined in Figure 3. This definition reflects the normal operation of these machines without the presence of any errors (i.e. stop signals). The conveyor-on and conveyor-off actions in the Hermes specification document are modelled here for simplicity as silent internal actions, τ. This is due to the fact that these actions cannot be interrupted and therefore do not constitute any externally observable behaviour. Running the full system, *System*, corresponds to running the two machines in parallel:

$$\textit{System} \stackrel{\text{def}}{=} (\textit{UP} \mid \textit{DP})$$

An important property of this protocol is to ensure that both machines agree on whether they have completed the PCB transfer successfully. We express this property as a predicate on the result of analysing the protocol.

Property 1 (Agreement on PCB Transfer Completion): Both machines, upstream and downstream, agree on the PCB transfer completion iff the following predicate, $\digamma_1$, defined on the result of analysing the protocol, ϕ, is true:

$\digamma_1(\phi) = \ \exists x \in \textit{bn}(\textit{UP}), y \in \textit{bn}(\textit{DP}) :$

$\quad$ "*Complete*" $\in \phi(x) \ \Leftrightarrow$ "*Complete*" $\in \phi(y)$ $\square$

This property captures the requirement that if one machine believes it has completed the transfer of the PCB, the other does as well, in whichever order that completion is signalled. Applying our static analysis, $\mathcal{A}[\![\textit{System}]\!]\phi_0 = \phi$, to the protocol specification of Figure 3, we obtain the following two sub-environments:

$$\phi = \phi_1[y' \mapsto \text{"}\textit{Complete}\text{"}, x \mapsto y'] \ \cup \ \phi_2[x' \mapsto \text{"}\textit{Complete}\text{"}, y \mapsto x']$$

where both ϕ_1 and ϕ_2 satisfy the above PCB transfer completion predicate $\digamma_1$. In this case, we did not need to use any approximation function since the set of names used is finite by nature due to the lack of any infinite (replicated) behaviour in the protocol's definition.

V. Error Tests

The definition of the Hermes protocol presented in Figure 3 of the previous section assumes normal operation and therefore does not cater for situations when errors are detected by any of the two machines, which could result in stopping their operation. Hence, we define in this section an enhanced formal specification that can handle such transport errors along the lines of the scenarios captured in [1, §2.3.4].

Before we can do that, we need to define a mechanism in the formal protocol specification to express the ability of the Hermes protocol in testing for the presence of such errors. One such mechanism for defining these *tests* can be specified using the condition evaluation construct in the π-calculus language:

$$\textit{state}(e_i).P_{e_i}[e_i = \text{"}\textit{stop}\text{"}]Q_{e_i}$$

The communication channel, *state*, is used to receive from the environment of the two machines any "*stop*" signals sent to those machines whenever transport errors occur in them. We use the enumerations, $i \in \mathbb{N}$, to distinguish the different copies of the input parameter, e, that receives such "*stop*" signals from the environment. Process P_{e_i} will run if a "*stop*" signal is received, otherwise Q_{e_i} runs if the received signal is different from "*stop*". We call P_{e_i} the *error-handling process*.

For brevity, we use the following shorthand representation of the above tests:

$$[e_i](Q_{e_i}) \ \equiv \ \textit{state}(e_i).P_{e_i}[e_i = \text{"}\textit{stop}\text{"}]Q_{e_i}$$

As a matter of fact, $[e_i]$ becomes an *alias* to uniquely reference the communication channel, *state*, input parameter, e_i, and the process, P_{e_i}. We stress here that these tests are part of the operation of the protocol itself, not a method to test the protocols behaviour.

The next question is *where should such tests be positioned in the protocol's specification?* The Hermes protocol [1, §2.3.4] provides only a limited number of points at which such tests are carried out (see the seven Scenarios U1a–D3). In our case, we consider that any sequential composition point with an action $a \in \{x(y), \overline{x}\langle y\rangle, \tau\}$ is a potential testing point that the protocol could use to detect errors in machines:

$$a \overset{\textit{test point}}{\frown} P$$

The Upstream Machine Process:

$$UP \stackrel{\text{def}}{=} \overline{machineready}().\overline{boardavailable}\langle\rangle.UP_{cont} \; + \; \overline{boardavailable}\langle\rangle.machineready().UP_{cont}$$

where,

$$UP_{cont} \stackrel{\text{def}}{=} starttransport().\tau.(\tau.\overline{transportfinished}\langle\text{“}Complete\text{”}\rangle.stoptransport(x).\mathbf{0} + stoptransport(x').\tau.\overline{transportfinished}\langle x'\rangle.\mathbf{0})$$

The Downstream Machine Process:

$$DP \stackrel{\text{def}}{=} \overline{machineready}\langle\rangle.boardavailable().DP_{cont} \; + \; boardavailable().\overline{machineready}\langle\rangle.DP_{cont}$$

where,

$$DP_{cont} \stackrel{\text{def}}{=} \tau.\overline{starttransport}\langle\rangle.(transportfinished(y').\tau.\overline{stoptransport}\langle y'\rangle.\mathbf{0} + \tau.\overline{stoptransport}\langle\text{“}Complete\text{”}\rangle.transportfinished(y).\mathbf{0})$$

Fig. 3. A π-calculus model of the Hermes v1.2 protocol in normal operation.

This implies that each machine should be able to be stopped *after* each action it performs, allowing for better range of detection of errors at more points in its operation.

Based on this, we redefine the formal specification of the Hermes protocol, as in Figure 4, in order to incorporate the ability to receive stop signals from the environment. The full system, *System*, will now consist of the two machines running in parallel with the environment, *Env*, as follows:

$$System \stackrel{\text{def}}{=} (UP \mid DP \mid Env)$$

The environment process, *Env*, will always send either a “*continue*” message, when there is no error to be signalled to either machine, or a “*stop*” message, when some transport error is detected. As a result, the definitions of *UP* and *DP* need to be modified, from their normal definitions in Figure 3, to be able to detect “*stop*” messages and to react to any error-handling behaviour such messages may initiate.

The specification of Figure 4 considers all possible points in a process where an error can be tested, once that process has commenced running (i.e. it has fired its first action). We identified 24 such points, compared to the original 7 points that were identified in the scenarios of [1, §2.3.4]. The error-testing specification also modifies the normal operation specification of Figure 3 by adding extra communication channels to cater for the RevokeMachineReady and RevokeBoardAvailable signals resulting from transport errors. Both *UP* and *DP* deviate to their error-handling processes once *“stop”* is received.

The individual error-handling processes, P_{e_i}, are defined in Figure 5 for the upstream machine, and Figure 6 for the downstream machine. We also indicate which of the scenarios in [1] each error-handling process corresponds to, or none if it does not correspond to any. An important observation that one can make here is that whilst the specification of [1] considers only scenarios where errors occur in either machine, the specification does not consider the possibility that errors may occur simultaneously in both machines. That is, the environment of Figure 4 could “*stop*” both machines at the same time. Due to the execution sequences of *UP* and *DP*, this possibility is limited to the following pairs of tests:

$([e_1],[e'_1]),([e_2],[e'_2]),([e_3],[e'_3]),([e_4],[e'_4]),([e_4],[e'_5]),$
$([e_2],[e'_5]),([e_5],[e'_6]),([e_6],[e'_6]),([e_7],[e'_6]),([e_5],[e'_{10}]),$
$([e_6],[e'_{10}]),([e_7],[e'_{10}]),([e_8],[e'_7]),([e_8],[e'_8]),([e_9],[e'_9]),$
$([e_{10}],[e'_{11}]),([e_{11}],[e'_{11}]),([e_{12}],[e'_{12}])$

In most of these pairs, nothing will occur as the neither point will offer actions that can synchronise with the other point in the pair. However, in a few cases, we find that certain error-handling processes may communicate with each other if both are triggered at the same time, as their actions can synchronise. This is the case for the following four pairs:
$([e_8],[e'_7]),([e_8],[e'_8]),([e_{10}],[e'_{11}]),([e_{11}],[e'_{11}])$

Next, we apply our static analysis, $\mathcal{A}([\![System]\!]\phi_0 = \phi'$, to the above version of the protocol specification with error tests. As a result, we obtain the following environment:

$$\begin{aligned}\phi' = (&\phi'_1[x'' \mapsto y', y' \mapsto \text{“}NotStarted\text{”}] \cup\\ &\phi'_2[x''' \mapsto y', y' \mapsto \text{“}Incomplete\text{”}] \cup\\ &\phi'_3[x'''' \mapsto y', y' \mapsto \text{“}Incomplete\text{”}] \cup\\ &\phi'_4[x' \mapsto \text{“}Incomplete\text{”}, y'' \mapsto x'] \cup\\ &\phi'_5[x' \mapsto \text{“}Incomplete\text{”}, y''' \mapsto x'] \cup\\ &\phi'_6[x \mapsto y', y' \mapsto \text{“}Complete\text{”}] \cup\\ &\phi'_7[x''''' \mapsto y', y' \mapsto \text{“}Complete\text{”}] \cup\\ &\phi'_8[y \mapsto x', x' \mapsto \text{“}Complete\text{”}] \cup\\ &\phi'_9[y'''' \mapsto x', x' \mapsto \text{“}Complete\text{”}] \cup\\ &\phi'_{10}[x''''' \mapsto \text{“}Complete\text{”}, y' \mapsto \text{“}Complete\text{”}] \cup\\ &\phi'_{11}[y'''' \mapsto \text{“}Complete\text{”}, x' \mapsto \text{“}Complete\text{”}] \cup\\ &\phi'_{12}[x \mapsto \text{“}Complete\text{”}, y' \mapsto \text{“}Complete\text{”}])\end{aligned}$$

Each sub-environment part of ϕ' represents a different path of execution, depending sometimes on which error points are triggered. This brings us to the following two properties.

Property 2 (Agreement on PCB Transfer Not Started): Both machines, upstream and downstream, agree that the PCB transfer has not started iff F_2 is true on the result of the analysis of the protocol, ϕ', where:

The Upstream Machine Process:

$$UP \stackrel{\text{def}}{=} \textit{machineready}().([e_1](\overline{\textit{boardavailable}}\langle\rangle.([e_2](UP_{cont}) + \textit{revokemachineready}().\mathbf{0})) + \overline{\textit{boardavailable}}\langle\rangle.[e_3](\textit{machineready}().([e_4](UP_{cont}) + \textit{revokemachineready}().\mathbf{0})) + \textit{revokemachineready}().\mathbf{0})$$

where,

$$UP_{cont} \stackrel{\text{def}}{=} \textit{starttransport}().[e_5](\tau.[e_6](\tau.[e_7](\overline{\textit{transportfinished}}\langle\text{“}Complete\text{”}\rangle.[e_8](\textit{stoptransport}(x).[e_9](\mathbf{0}))) + \textit{stoptransport}(x').[e_{10}](\tau.[e_{11}](\overline{\textit{transportfinished}}\langle x'\rangle.[e_{12}](\mathbf{0}))))))$$

The Downstream Machine Process:

$$DP \stackrel{\text{def}}{=} \overline{\textit{machineready}}\langle\rangle.[e'_1](\textit{boardavailable}().([e'_2](DP_{cont}) + \textit{revokeboardavailable}().\mathbf{0})) + \textit{boardavailable}().([e'_3](\overline{\textit{machineready}}\langle\rangle.([e'_4](DP_{cont}) + \textit{revokeboardavailable}().\mathbf{0}) + \textit{revokeboardavailable}().\mathbf{0}))$$

where,

$$DP_{cont} \stackrel{\text{def}}{=} \tau.[e'_5](\overline{\textit{starttransport}}\langle\rangle.[e'_6](\textit{transportfinished}(y').[e'_7](\tau.[e'_8](\overline{\textit{stoptransport}}\langle y'\rangle.[e'_9](\mathbf{0}))) + \tau.[e'_{10}](\overline{\textit{stoptransport}}\langle\text{“}Complete\text{”}\rangle.[e'_{11}](\textit{transportfinished}(y).[e'_{12}](\mathbf{0})))))$$

The Test Environment Process:

$$Env \stackrel{\text{def}}{=} (!\ \overline{\textit{state}}\langle\text{“}\textit{continue}\text{”}\rangle.\mathbf{0}) \mid (!\ \overline{\textit{state}}\langle\text{“}\textit{stop}\text{”}\rangle.\mathbf{0})$$

Fig. 4. A formal model of the Hermes v1.2 protocol with error tests.

Process Definition	**Description**
$P_{e_1} \stackrel{\text{def}}{=} \mathbf{0}$	*No corresponding scenario*
$P_{e_2} \stackrel{\text{def}}{=} \overline{\textit{revokeboardavailable}}\langle\rangle.\mathbf{0}$	*Corresponds to Scenario U1a*
$P_{e_3} \stackrel{\text{def}}{=} P_{e_2}$	*No corresponding scenario*
$P_{e_4} \stackrel{\text{def}}{=} P_{e_2}$	*No corresponding scenario*
$P_{e_5} \stackrel{\text{def}}{=} \overline{\textit{transportfinished}}\langle\text{“}\textit{NotStarted}\text{”}\rangle.\ \textit{stoptransport}(x'').\mathbf{0}$	*Corresponds to Scenario U1b*
$P_{e_6} \stackrel{\text{def}}{=} \tau.\overline{\textit{transportfinished}}\langle\text{“}\textit{Incomplete}\text{”}\rangle.\ \textit{stoptransport}(x''').\mathbf{0}$	*Corresponds to Scenario U2*
$P_{e_7} \stackrel{\text{def}}{=} \overline{\textit{transportfinished}}\langle\text{“}\textit{Incomplete}\text{”}\rangle.\ \textit{stoptransport}(x'''').\mathbf{0}$	*No corresponding scenario*
$P_{e_8} \stackrel{\text{def}}{=} \textit{stoptransport}(x''''').\mathbf{0}$	*Corresponds to Scenario U3*
$P_{e_9} \stackrel{\text{def}}{=} \mathbf{0}$	*No corresponding scenario*
$P_{e_{10}} \stackrel{\text{def}}{=} \tau.\overline{\textit{transportfinished}}\langle\text{“}\textit{Complete}\text{”}\rangle.\mathbf{0}$	*No corresponding scenario*
$P_{e_{11}} \stackrel{\text{def}}{=} \overline{\textit{transportfinished}}\langle\text{“}\textit{Complete}\text{”}\rangle.\mathbf{0}$	*No corresponding scenario*
$P_{e_{12}} \stackrel{\text{def}}{=} \mathbf{0}$	*No corresponding scenario*

Fig. 5. Definition of error-handling processes for the upstream machine.

Process Definition	Description
$P_{e'_1} \stackrel{\text{def}}{=} \overline{\textit{revokemachineready}}\langle\rangle.\mathbf{0}$	*No corresponding scenario*
$P_{e'_2} \stackrel{\text{def}}{=} P_{e'_1}$	*Corresponds to Scenario D1*
$P_{e'_3} \stackrel{\text{def}}{=} \mathbf{0}$	*No corresponding scenario*
$P_{e'_4} \stackrel{\text{def}}{=} P_{e'_1}$	*No corresponding scenario*
$P_{e'_5} \stackrel{\text{def}}{=} \tau.P_{e'_1}$	*No corresponding scenario*
$P_{e'_6} \stackrel{\text{def}}{=} \tau.\overline{\textit{stoptransport}}\langle\text{“}\textit{Incomplete}\text{”}\rangle.$ $\textit{transportfinished}(y'').\mathbf{0}$	*Corresponds to Scenario D2*
$P_{e'_7} \stackrel{\text{def}}{=} \tau.\overline{\textit{stoptransport}}\langle\text{“}\textit{Complete}\text{”}\rangle.\mathbf{0}$	*Corresponds to Scenario D3*
$P_{e'_8} \stackrel{\text{def}}{=} \overline{\textit{stoptransport}}\langle\text{“}\textit{Complete}\text{”}\rangle.\mathbf{0}$	*No corresponding scenario*
$P_{e'_9} \stackrel{\text{def}}{=} \mathbf{0}$	*No corresponding scenario*
$P_{e'_{10}} \stackrel{\text{def}}{=} \overline{\textit{stoptransport}}\langle\text{“}\textit{Incomplete}\text{”}\rangle.$ $\textit{transportfinished}(y''').\mathbf{0}$	*No corresponding scenario*
$P_{e'_{11}} \stackrel{\text{def}}{=} \textit{transportfinished}(y'''').\mathbf{0}$	*No corresponding scenario*
$P_{e'_{12}} \stackrel{\text{def}}{=} \mathbf{0}$	*No corresponding scenario*

Fig. 6. Definition of error-handling processes for the downstream machine.

$$\digamma_2(\phi') = \exists x \in \textit{bn}(\textit{UP}), y \in \textit{bn}(\textit{DP}) : \text{“}\textit{Not Started}\text{”} \in \phi'(x) \Leftrightarrow \text{“}\textit{Not Started}\text{”} \in \phi'(y)\ \square$$

$\digamma_2$ is satisfied by the ϕ'_1 sub-environment. The $\digamma_2$ predicate corresponds to error-handling in Scenario U1b in [1], where the error occurs in the upstream machine before it starts transporting the PCB. On the other hand, the next property captures all errors that occur in both machines while the PCB is in the middle of being transported.

Property 3 (Agreement on PCB Transfer Not Completed): Both machines, upstream and downstream, agree that the PCB transfer has not been completed iff $\digamma_3$ is true, where:

$$\digamma_3(\phi') = \exists x \in \textit{bn}(\textit{UP}), y \in \textit{bn}(\textit{DP}) : \text{“}\textit{Incomplete}\text{”} \in \phi'(x) \Leftrightarrow \text{“}\textit{Incomplete}\text{”} \in \phi'(y)\ \square$$

$\digamma_3$ is satisfied by ϕ'_2 – ϕ'_5 sub-environments. Of these, ϕ'_2 corresponds to Scenario U2 and ϕ'_4 corresponds to Scenario D2 in [1]. Both ϕ'_3 and ϕ'_5 have no corresponding scenarios and are therefore, a new result of this analysis.

It is also worth noting now that the PCB transfer completion is satisfied not only due to the $\digamma_1$ predicate, which was satisfied by the $\phi_1 = \phi'_6$ and $\phi_2 = \phi'_8$ environments, but also due to the additional cases of ϕ'_7, ϕ'_9, ϕ'_{10}, ϕ'_{11} and ϕ'_{12}. In particular, the cases of ϕ'_{10} and ϕ'_{11} are a result of the communications between the pairs $([e_8],[e'_7]), ([e_8],[e'_8]), ([e_{10}],[e'_{11}])$ and $([e_{11}][e'_{11}])$ as indicated earlier. The case of ϕ'_7 corresponds to Scenario U3 and the case of ϕ'_{12} to Scenario D3 [1]. However, ϕ'_9 is a new case that does not correspond to any of the standard scenarios. We now define *robustness* as the result that the two machines always agree on the same status in any environment.

Lemma 1: The Hermes protocol is robust since $\digamma_1 \wedge \digamma_2 \wedge \digamma_3$.

Proof. $\digamma_1 \wedge \digamma_2 \wedge \digamma_3$ can be shown to be true from Properties 1–3. $\square$

VI. Conclusion

To conclude the paper, we have demonstrated that despite the lack of testing of some parts of the Hermes protocol in its original specification document [1], and the lack of consideration for simultaneous errors, the protocol behaves in the intended manner. We used a formal specification and analysis approach to demonstrate the robustness of the protocol by demonstrating that the protocol behaves as expected leading to upstream and downstream machine agreement in every case where the transfer completes successfully, due to lack of errors, or does not start or complete, due to presence of errors at some stage of the board transfer process.

As part of future work, we plan to model and analyse other Industry 4.0 communication technologies [46], e.g. the Open Platform Communications (OPC) Unified Architecture [27], Bosch's Production Performance Management Protocol (PPMP) [47] and the Data Distribution Service [48]. We also plan to specify probabilistic and stochastic properties and verify whether such properties hold in the Hermes protocol and other Industry 4.0 protocols.

Another interesting area to extend this research to would be the new IPC-CFX (Connected Factory Exchange) standard [49], which transforms a Hermes-based manufacturing line into a fully Industrial Internet-of-Things environment. As far as we know, the CFX standard is not yet made open-source.

References

[1] T. H. S. Initiative, "IPC-HERMES-9852: The global standard for machine-to-machine communication in SMT assembly (version 1.2)," IPC, Tech. Rep., 2019.

[2] IPC, "IPC-SMEMA-9851: Mechanical Equipment Interface Standard," IPC - Association Connecting Electronics Industries, Tech. Rep., 2007.

[3] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes," *Information and Computation*, vol. 100(1), pp. 1–77, Sep. 1992.

[4] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[5] L. M. Barroca and J. A. McDermid, "Formal methods: Use and relevance for the development of safety-critical systems," *The Computer Journal*, vol. 35, no. 6, pp. 579–599, 1992.

[6] C. J. Burgess, "The role of formal methods in software engineering education and industry," University of Bristol, Bristol, UK, UK, Tech. Rep., 1995.

[7] P. G. Larsen, J. Fitzgerald, and T. Brookes, "Applying formal specification in industry," *IEEE software*, vol. 13, no. 3, pp. 48–56, 1996.

[8] P. Wegner, "The vienna definition language," *ACM Computing Surveys*, vol. 4, no. 1, pp. 5–63, Mar. 1972.

[9] J. M. Spivey, *Understanding Z: A Specification Language and Its Formal Semantics*. New York, NY, USA: Cambridge University Press, 1988.

[10] J.-R. Abrial, *The B-book: Assigning Programs to Meanings*. New York, NY, USA: Cambridge University Press, 1996.

[11] D. Craigen, S. Gerhart, and T. Ralston, "An international survey of industrial applications of formal methods," in *Z User Workshop, London 1992*. Springer, 1993, pp. 1–5.

[12] M. G. Hinchey and J. P. Bowen, *Applications of formal methods*. Prentice Hall New Jersey, 1995, vol. 1.

[13] J. Bowen and V. Stavridou, "Safety-critical systems, formal methods and standards," *Software Engineering Journal*, vol. 8, no. 4, pp. 189–209, 1993.

[14] J.-R. Abrial, E. Börger, H. Langmaack *et al.*, *Formal methods for industrial applications: Specifying and programming the steam boiler control*. Springer Science & Business Media, 1996, vol. 9.

[15] G. Frey and L. Litz, "Formal methods in plc programming," in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, vol. 4. IEEE, 2000, pp. 2431–2436.

[16] A. Banks and R. Gupta, "MQ Telemetry Transport (MQTT) V3.1.1 Protocol Specification: Committee Specification Draft 02 / Public Review Draft 02," IBM Corporation, Tech. Rep., 2014.

[17] B. Aziz, "A Formal Model and Analysis of the MQ Telemetry Transport Protocol," in *9th International Conference on Availability, Reliability and Security (ARES 2014), Fribourg, Switzerland*. IEEE, 2014, pp. 59–68.

[18] B. Aziz, "A formal model and analysis of an IoT protocol," *Ad Hoc Networks*, vol. 36, pp. 49–57, 2016.

[19] S. Chouali, A. Boukerche, and A. Mostefaoui, "Towards a formal analysis of mqtt protocol in the context of communicating vehicles," in *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*. ACM, 2017, pp. 129–136.

[20] K. Mladenov, "Formal verification of the implementation of the MQTT protocol in IoT devices," Master's thesis, University of Amsterdam, the Netherlands, 2017.

[21] M. Houimli, L. Kahloul, and S. Benaoun, "Formal specification, verification and evaluation of the mqtt protocol in the internet of things," in *2017 International Conference on Mathematics and Information Technology (ICMIT)*. IEEE, 2017, pp. 214–221.

[22] C. Bormann, A. P. Castellani, and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[23] K. İnçki and I. Ari, "A novel runtime verification solution for iot systems," *IEEE Access*, vol. 6, pp. 13 501–13 512, 2018.

[24] R. Kowalski and M. Sergot, "A logic-based calculus of events," *New Generation Computing*, vol. 4, no. 1, pp. 67–95, Mar 1986.

[25] A. J. Vattakunnel, N. S. Kumar, and G. S. Kumar, "Modelling and verification of coap over routing layer using spin model checker," *Procedia Computer Science*, vol. 93, pp. 299–308, 2016.

[26] M. Puys, M.-L. Potet, and P. Lafourcade, "Formal analysis of security properties on the opc-ua scada protocol," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2016, pp. 67–75.

[27] "The OPC Unified Architecture," https://opcfoundation.org/about/opc-technologies/opc-ua/, accessed: 11-06-2019.

[28] "ProVerif: Cryptographic protocol verifier in the formal model," https://prosecco.gforge.inria.fr/personal/bblanche/proverif/, accessed: 11-06-2019.

[29] S. Rohjans, K. Piech, and S. Lehnhoff, "Uml-based modeling of opc ua address spaces for power systems," in *2013 IEEE International Workshop on Inteligent Energy Systems (IWIES)*, Nov. 2013, pp. 209–214.

[30] P. Herrmann and J. O. Blech, "Formal model-based development in industrial automation with reactive blocks," in *Federation of International Conferences on Software Technologies: Applications and Foundations*. Springer, 2016, pp. 253–261.

[31] "Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure," https://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/, accessed: 03-06-2019.

[32] "Bitreactive AS: Reactive Blocks," http://www.bitreactive.com, accessed: 03-06-2019.

[33] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Tool support for the rapid composition, analysis and implementation of reactive services," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2068 – 2080, 2009.

[34] M. Rausch and H. . Hanisch, "Net condition/event systems with multiple condition outputs," in *Proceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, vol. 1, Oct. 1995, pp. 592–600 vol.1.

[35] H.-M. Hanisch, A. Lobov, J. L. M. Lastra, R. Tuokko, and V. Vyatkin, "Formal validation of intelligent-automated production systems: towards industrial applications," *International Journal of Manufacturing Technology and Management*, vol. 8, no. 1-3, pp. 75–106, 2006.

[36] S. Zahra, M. Alam, Q. Javaid, A. Wahid, N. Javaid, S. U. R. Malik, and M. Khurram Khan, "Fog computing over iot: A secure deployment and formal verification," *IEEE Access*, vol. 5, pp. 27 132–27 144, Nov. 2017.

[37] "Shibboleth," https://www.shibboleth.net/, accessed: 11-06-2019.

[38] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[39] K. Jensen, "High-level petri nets," in *Applications and Theory of Petri Nets*. Springer, 1983, pp. 166–180.

[40] D. Chen and G. Chang, "A survey on security issues of m2m communications in cyber-physical systems." *KSII Transactions on Internet & Information Systems*, vol. 6, no. 1, 2012.

[41] B. Aziz, "Measuring the speed of information leakage in mobile processes," in *Proceedings of the* 11^{th} *International Conference on Algebraic Methodology and Software Technology*, ser. Lecture Notes in Computer Science, vol. 4019. Kuressaare, Estonia: Springer Verl formag, Jul. 2006, pp. 36–50.

[42] B. Aziz, "A static analysis framework for security properties in mobile and cryptographic systems," Ph.D. dissertation, School of Computing, Dublin City University, Dublin, Ireland, 2003.

[43] D. Sangiorgi and D. Walker, *The Pi-Calculus - A Theory of Mobile Processes*. Cambridge, UK: Cambridge University Press, 2001.

[44] B. Aziz and G. Hamilton, "A Privacy Analysis for the π-calculus: The Denotational Approach," in *Proceedings of the* 2^{nd} *Workshop on the Specification, Analysis and Validation for Emerging Technologies*, ser. Datalogiske Skrifter, no. 94. Copenhagen, Denmark: Roskilde University, Jul. 2002.

[45] B. Aziz, G. Hamilton, and D. Gray, "A static analysis of cryptographic processes: The denotational approach," *Journal of Logic and Algebraic Programming*, vol. 64(2), pp. 285–320, Aug. 2005.

[46] P. Marcon, F. Zezulka, I. Vesely, Z. Szabo, Z. Roubal, O. Sajdl, E. Gescheidtova, and P. Dohnal, "Communication technology for industry 4.0," in *2017 Progress In Electromagnetics Research Symposium - Spring (PIERS)*, May 2017, pp. 1694–1697.

[47] "The PPMP Specification," https://www.eclipse.org/unide/specification/, accessed: 11-06-2019.

[48] "The Data Distribution Service Foundation," https://www.dds-foundation.org/, accessed: 11-06-2019.

[49] C. F. Initiative, "IPC-2591: Connected Factory Exchange (CFX)," IPC, Tech. Rep., 2019.

Benjamin Aziz is a Senior Lecturer at the School of Computing, University of Portsmouth. Benjamin holds PhD degree in formal verification of computer security from Dublin City University (2003) and has research interests and experience in the field of computer and information security, with over 130 publications related to areas such as security engineering of large-scale systems, IoT and SDN security, formal analysis, requirements engineering and digital forensics. He is on board several program committees for international conferences and working groups, including ERCIM's FMICS, STM, Cloud Security Alliance and IFIP WG11.3.