A JSON/HTTP communication protocol to support the development of distributed cyber-physical systems

Fernando Pereira[*±†], Luís Gomes [±†]
[±]*Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia – Portugal*
[*]*ISEL, Instituto Superior de Engenharia de Lisboa- Portugal*
[†] *UNINOVA - CTS - Portugal*
fjp@deea.isel.ipl.pt, lugo@fct.unl.pt

Abstract - Cyber physical systems are often built using networks of components containing physical and computational resources, enabling the design of distributed applications that collect data from remote sensors and manipulate remote actuators, located on local networks or on far away locations. The communication protocol presented in this paper was designed to support the communication between components of distributed applications. In addition, it includes remote debug and monitoring capabilities, to support the detection and resolution of errors and design mistakes on nodes running on remote locations. Employing the JSON/HTTP standards, the proposed protocol is Web browser friendly, suitable for the creation of Web based applications and user interfaces, but it may be employed on most programming environments that offer libraries to support those standards. As it is based on HTTP, it can easily traverse most firewall configurations and used through proxies. Development of the proposed protocol started on the IOPT-Tools framework, but the current version was implemented as part of the IOPT-Flow framework, aiming the development of distributed CPS applications based on graphical formalism combining Petri nets and dataflows. Both tool frameworks are available at http://gres.uninova.pt.

I. Introduction

The field of cyber-physical systems (CPS) deals with the interaction between physical and computational components with focus on distribution and communication aspects [1]. A wide range of CPS applications can be found in the literature [2][3][4], including industrial systems, smart grids, traffic control systems, wireless sensor networks, medical systems, vending machines and entertainment and home appliances, among many others.

Extending the concepts inherited from embedded systems, cyber-physical systems take advantage of the existing Internet infrastructure to enabled the design of sophisticated applications. For example, controller software running on the cloud can collect vast amounts of information from networks of remote sensors and employ big-data and artificial intelligence techniques to optimize production or predict future device failures. Industrial applications often extend beyond the reach of a single company and may collect data from multiple partners of a vertical supply chain to support just-in-time production management.

The development of cyber-physical systems is considered a multidisciplinary area that may require the collaboration from persons from different technical and scientific backgrounds, including software development, mechatronics, mechanical and electrical engineering, and knowledge about specific areas related to each application. As a consequence, people from various backgrounds may employ different concepts, use different terminology, and may not have training about software development technologies.

The IOPT-Flow Web based modeling framework [5] was designed to support the development of distributed cyber-physical systems using a graphical modeling formalism that may appeal to persons from different backgrounds called DS-Pnets (Dataflows, Signals and Petri nets). Combining Petri nets [6][7] and dataflows [8][9], the new formalism may be used to design mixed controllers containing both state machines that react to external events and data processing operations used for signal processing and conditioning, data filtering and apply mathematical transformations.

Model composition is performed using components that may be located on different remote nodes. Each component communicates with the external world using an interface composed by input and output signals and events. This way, entire distributed applications are created just by collecting multiple components from different nodes and connecting the respective inputs and outputs using arcs.

Automatic code generation tools convert the graphical models to multiple programming languages, implementing the behavior of each component on C/C++, JavaScript [10] or VHDL [11], including the required JSON/HTTP [12] client/server code for each node, automating the implementation of distributed CPS applications without the need to manually write low-level software/hardware code.

Although the default formalism used for component design is the DS-Pnet graphical language [13], is it also possible to employ other languages as IOPT nets from the parent IOPT-Tools framework [14] and traditional programming languages. A single application may mix components designed using different languages and reuse existing software objects and algorithms. The code written in traditional languages is encapsulated inside "foreign" components that may be used inside applications just by adding the respective component symbol and drawing arcs, in the same way as native DS-Pnet components. This way, the software developers writing foreign components do not need to care with any communication details, that continue to be dealt by the automatic code generation infrastructure.

The communication protocol presented in this paper plays a central role in the resulting applications, allowing the usage of publisher-subscriber and client-server usage patterns

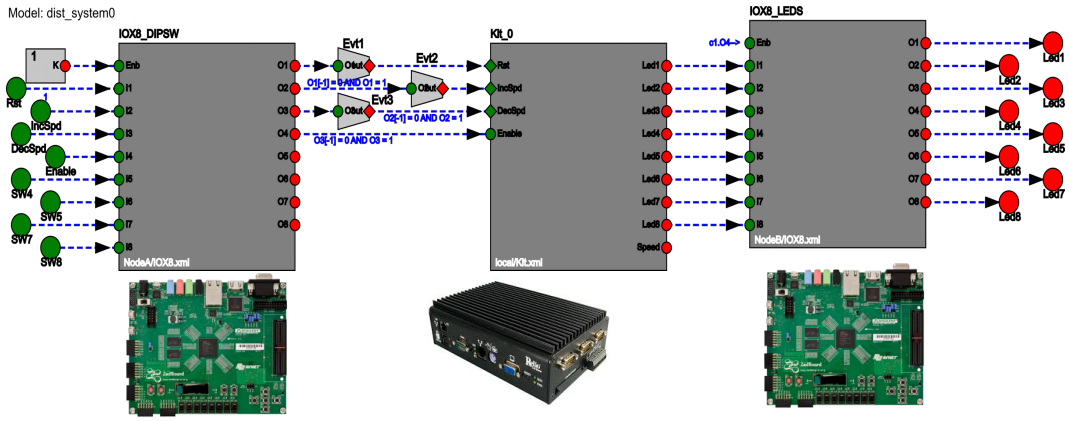

Fig. 1: Distributed DS-Pnet model implemented on 3 nodes (1 embeded PC and 2 Zed boards)

employed to establish the the interconnection between remote components. In addition to inter-component machine-to-machine communication, the proposed protocol offers remote debug and monitoring capabilities and may be used the design Web based graphical user interfaces.

An existing remote debugger application employs this protocol to connect to remote nodes running the automatically generated code and monitor the graphical models deployed on those devices, with capabilities for step-by-step trace execution, define breakpoints and collect signal waveforms. Remote debugging is particularly important as the CPS components may be located far-away, placed on inaccessible points, or implemented on low end IoT (Internet of Things) devices lacking user interface hardware.

II. DS-Pnets

The DS-Pnet graphical modeling formalism [13] employs the concept of components to permit the modular design of distributed cyber-physical system applications. Like the parent IOPT-net class[7], each component has a external interface composed by input and output signals and events. Component inputs may be directly driven by physical signals, or may be driven by outputs of other components. In the same way, component outputs may be connected to physical actuators, or used to drive the inputs of other components, either located in the same node or on remote locations.

Figure 1 presents a very simple example of a DS-Pnet application containing three components running on different nodes. The communication between the components is specified using arcs connecting the respective input and output signals and events. The green circles on the left correspond to input signals associated to physical pins that are connected to sensors, buttons and switches on a Zed-board. In the same way, the red circles on the right correspond to output signals associated to physical pins on other board, used to drive actuators or LEDs. The central component runs on an embedded computer and contains only computational resources used to generate animated sequences of LED flashes. Three dataflow operations are used to detect events associated to logical transitions from 0 to 1 on input signals.

The behavior of each component is specified using DS-Pnet or IOPT-net models. For example, the model presented in figure 2 implements a UART transmitter component (to run on FPGAs using VHDL code generation tools [15]).

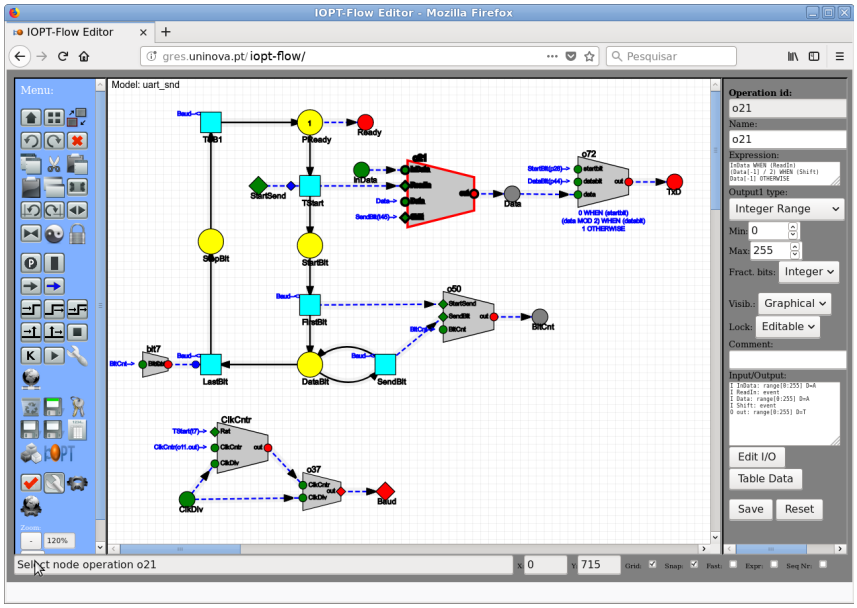

Fig. 2: The IOPT-Flow Web user interface

To model the behavior of system controllers, DS-Pnets offer a mix of Petri nets and dataflows. Petri nets, inherited from the parent IOPT Petri net class [7], are employed to design "state-machines" that control the system state evolution according to external events. Dataflow operations are used to apply mathematical transformations to input signals, perform digital signal processing and calculate the value of output signals. As can be viewed in figure 2, the dataflow operations interact with Petri net nodes, working as input events and guard conditions used to inhibit the firing of Petri net transitions. In the same way, the values of Petri net place marking and events triggered by transition firing may be used to perform dataflow computations.

The execution of a DS-Pnet model is performed in discrete steps where all dataflow operations and Petri net transitions are evaluated. A maximal step execution semantics is employed, meaning that all transitions enabled and ready to fire, will fire immediately on the next execution step.

In addition to native components, designed using DS-Pnet or IOPT-net models, it is also possible to create foreign components using traditional programming languages. Each class of foreign components must implement two functions, to initialize the component and run one execution step. The execution step function calculates the value of all component outputs according to the internal state variables and the value of the respective inputs.

III. The IOPT-Flow tool framework

Figure 2 displays the user interface of thee IOPT-Flow tool framework running on a Web browser. The tool-chain includes a graphical editor, a simulator, a remote debugger and a model checking subsystem.

The editor is used to design and edit models to create entire applications or new components. It contains an extensible library of previously designed components, containing native components (DS-Pnet / IOPT-net models) and foreign components (coded using external languages). To simplify the design of distributed applications the editor has the ability to import components form remote hardware devices that are running the code generated automatically and paste references to these components on new models.

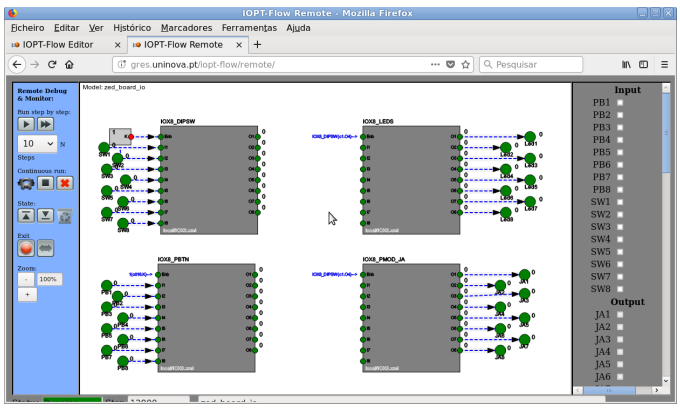

Fig. 3: Remote debugger tool

Models may be executed and debugged directly on the Web browser using the simulator tool, permitting the early detection of design mistakes, before reaching the prototype implementation phase and without the risk of damaging physical devices. It permits continuous running and step-by-step execution, break-point definition and store signal waveforms for posterior inspection (fig. 4). The remote debugger application, presented in figure 3, has a similar interface and provide equivalent features, but is used to debug models running on remote hardware.

The automatic code generation tools permit the execution of entire applications or just components on embedded hardware platforms using the C, VHDL[11] and JavaScript[10] languages. In addition to the model execution semantics, the software code generated automatically also contains a JSON/HTTP client/server infrastructure that automates the creation of distributed applications.

An IOPT model-checking subsystem [16], used to analyze the Petri net part of DS-Pnet models, including a state-space generator and a query system that automates the analysis of large state-space graphs is used to detect potential deadlock and live-lock situations and the reach-ability of undesirable states corresponding to safety rule violations and hardware malfunctions.

Finally, a library of foreign components is used to extend the functionality of the core language, adding arrays and matrices, file input and output, system time, graphical/Web user interface widgets and an interface to communicate with industrial devices using the ModBUS field-bus. The user interface widgets may be used to automatically create graphical user interfaces to monitor and operate remote systems. It may be employed on PCs and embedded hardware using the C code generated automatically, or on Web browsers in association with the simulator tool.

IV. Related Work

The communication protocol presented in this paper was created in the context of the IOPT-Flow tool framework and was designed to permit the distributed execution of DS-Pnet

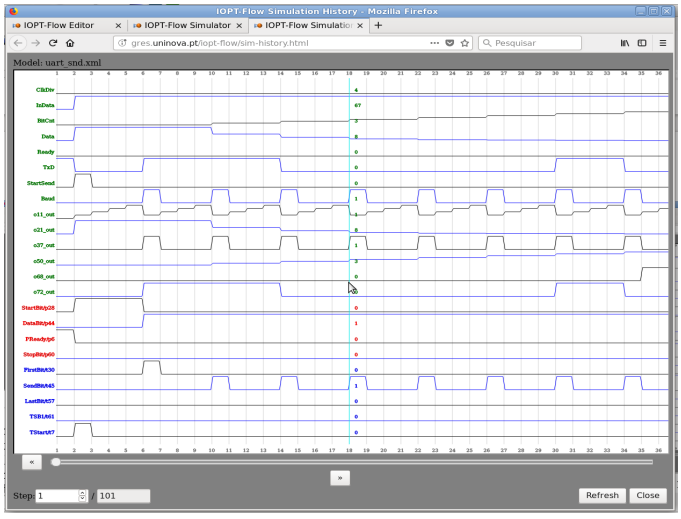

Fig. 4: Simulation Waveforms

models and provide remote debug and monitoring capabilities, including step-by-step execution and break-point definition. It implements a set of remote procedures that directly match the requirements of the code produced by the automatic code generation tools, supporting the communication between remote components and the enumeration of model meta-data to support the remote debugging and the editor tool and permit the insertion of components already running on embedded hardware into new models.

The DS-Pnet modeling language combines Petri nets and data-flows. Both of these formalisms have been extensively used in the past for the development of embedded systems and industrial automation systems. For example, the NCES/SNS, CPN, Renew, CPN-AMI Petri net classes [17][18][19][20] [21] and usage of synchronous data-flows in the the Ptolomy II project [22], should be mentioned.

The current version of the communication protocol is an evolution of a previous work presented in [23]. These protocols were used in association with the IOPT Tools framework, aiming the remote debug and monitoring of embedded system controllers and the creation of remote Web user interfaces. However, the previous protocol was thought for centralized controllers and did not to support distributed applications composed by networks of components located in different nodes.

A typical use-case of the previous protocol involves just a single client connected to a monolithic embedded application, supporting remote debug and monitoring and the creation of user interfaces for remote control and monitoring. In contrast, a typical use case of the new protocol involves multiple concurrent clients of distributed applications containing many components located on several networked nodes. The same node may contain components that are clients of other nodes and also serve requests from other nodes. In a single application, multiple nodes may implement user interfaces that allow the control and operation of remote physical devices located on different nodes. In the same way, a system designer

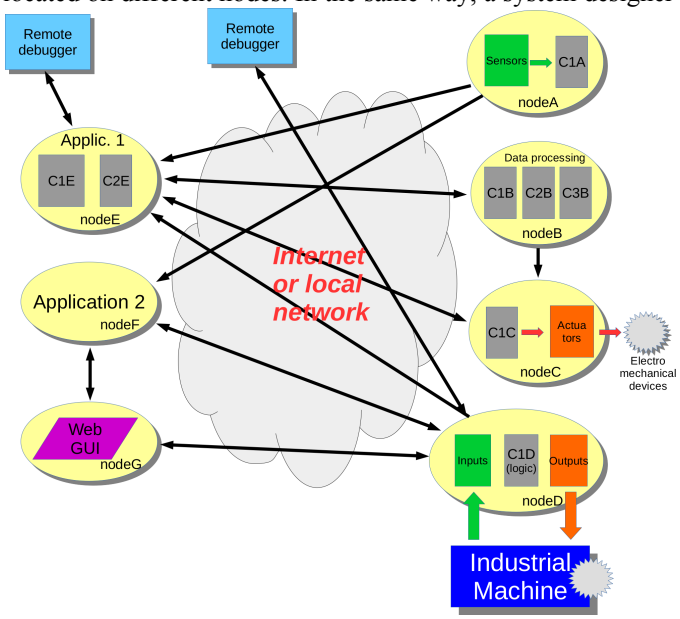

Fig. 5: Distributed CPS network topology example

may open multiple instances of the remote debugger application connected to different nodes of the application. As a result, the new protocol must answer different requirements, including support for components, enhanced user authentication and support for publisher/subscriber and client/server use-cases.

As the entire tool-chain was implemented using Web technologies and most tools run directly on Web browsers, the proposed protocol employs Web standards, JSON and HTTP, that are supported directly by all Web browsers, without the need of third party libraries. In contrast, other machine-to-machine protocols commonly used in Internet of things (IoT) [24], as MQTT [25], CoAP [26] and OPC/UA [27] employ binary data formats over UDP/IP or TCP/IP connections and do not use HTTP[1]. As a result, these protocols are usually filtered by default on most firewall configurations. In contrast, the proposed protocol employs HTTP, that is usually open and may be forwarded using standard HTTP proxies.

Although binary data formats offer advantages in terms of bandwidth consumption, the HTTP protocol can compensate this disadvantage using data compression. As the data payload is encoded in JSON format, where the same keywords are repeated many time, HTTP data compression can also bring great bandwidth savings, and compression is planned for a future protocol revision.

The procedure calls implemented by the new protocol could have been implemented on top of other protocol standards, as OPC/UA or CoAP, frequently used by IoT frameworks. However, most of the these frameworks also support JSON and HTTP and in the future it may be possible to communicate with DS-Pnet components from these frameworks. In alternative, future work may include the design of adapter software to obtain interoperability with devices supporting the MQTT/CoAP/OPC-UA protocols.

V. Communication Protocol

The new communication protocol was designed to support distributed CPS applications composed by networks of components located on multiple nodes. Figure 5 presents a possible network topology where two applications interact with remote nodes containing computational and physical devices. Some component inputs and outputs may be connected to physical devices (sensors and actuators) and other inputs and outputs are used to communicate with other components.

In this topology, each node runs the code produced by the IOPT-Flow automatic code generation tools to run a DS-Pnet/IOPT-net model that may contain one or more components. Usually the nodes associated to physical devices run the output of the C and VHDL code generators on embedded hardware platforms, the purely computational nodes may execute foreign components designed using other programming languages and the user interface nodes may be

1 Although CoAP may be forwared over HTTP proxies and a version of OPC/UA employs XML/HTTP, the most widely disseminated version is based on binary data formats.

executed directly on Web browsers using the output of the JavaScript code generator.

To permit communication, the output of the C code generator includes a minimalist HTTP server that is executed on each node and may receive connections from other nodes or from the remote debugger. In the same way, when the DS-Pnet model running on a node has arcs connected to other nodes, then HTTP client code is also added to the output of the C and JavaScript code generators.

As the CPS nodes communicate over the Internet and each node may hold multiple simultaneous client connections, the HTTP servers must deal with security issues.. As a result, a simple user authentication sub-system was created, supporting multiple users with different privilege levels according to the respective tasks, listed in the next table:

Level	Name	Description
0	Unauthorized	Connection not currently authenticated (default state before successful authentication)
1	Observer	May only read or subscribe signals and events and read internal system data (place marking, etc.), typically used to subscribe sensor information from public services.
2	Client	In addition to level 1, may also concurrently emit events and define input values
3	Master	In addition to level 2, may exclusively grab input signals and events and perform trace and debug operations: pause, run single steps and force input values. Used in typical distributed applications that establish a static network topology where a main model obtains exclusive rights over remote component inputs.
4	Administrator	In addition to level 3, may reconfigure the network, disconnecting components from existing network nodes and reconnecting to alternative nodes. May be used to create load balancing and fault tolerance solutions. (reserved for future implementations)

Table 1: Privilege levels

User authentication is performed using a challenge response mechanism based on cryptography keys, avoiding the transmission of passwords in clear text. Challenges are randomly generated for each connection, avoiding challenge repetition. Privilege levels are requested on a connection basis, meaning that different connections from the same user may hold different privilege levels.

Components are addressed using resource location strings, composed by a user-name, a node name and a component identifier: «http://user@address:port/component-id». An optional port number may be added (default 9000) and the address may consist of a text based Internet address or a numeric IP address. In alternative, the resource location may be specified using just a symbolic node name, that will be resolved at execution time from the information contained in a local file «node_db.txt». This file contains a list of symbolic node names that are associated with the definitive resource locations, permitting the redefinition of the node addresses during the system deployment, without the need to change models and recompile the generated code.

```
get http://host/login?user=guest HTTP/1.0
{"sess_key":"bd59c671049c7bb7fb4e109674cadfe2775b68fb"}
get http://host/requestPriv?priv=2&priority=5&sessid=65ae1dca6b7400b992b8b454c4b8952568f1bb88 HTTP/1.0
{"priv":2}
get http://host/getModelName?sessid=65ae1dca6b7400b992b8b454c4b8952568f1bb88 HTTP/1.0
{"model_name":"ui_test","version":"V1.0"}
get http://host/getModelURL?sessid=65ae1dca6b7400b992b8b454c4b8952568f1bb88 HTTP/1.0
{"model_url":"http://gres.uninova.pt/iopt-flow/files/ui_test.xml"}
get http://host/listModelMetadata&sessid=65ae1dca6b7400b992b8b454c4b8952568f1bb88
{"attributes":[
  {"name":"NV","node":"input","type":"int-range","min":0,"max":65535,"def-value":0},
  {"name":"Page","node":"input","type":"int-range","min":0,"max":15,"def-value":0},
{"name":"Sens","node":"input","type":"boolean","def-value":0},
{"name":"Vis","node":"input","type":"boolean", "on-error":1,"def-value":0},
{"name":"Checked","node":"output","type":"boolean","driven":1,"def-value":0},
{"name":"Disp","node":"output","type":"boolean","driven":1,"def-value":0},
  {"name":"Pct","node":"output","type":"int-range","min":0,"max":65535,"driven":1,"def-value":0},
{"name":"Status","node":"output","type":"boolean","driven":1,"def-value":0},
  {"name":"Val","node":"output","type":"int-range","min":0,"max":65535,"driven":1,"def-value":0},
{"name":"R","node":"input","type":"event","on_error":0,"def-value":0},
{"name":"S","node":"input","type":"event","on_error":0,"def-value":0},
{"name":"ChgEvt","node":"output","type":"event","driven":1,"on_error":0,"def-value":0}]
}
```

Listing 1

The availability of model meta-data plays an important role in the design of new applications employing components that have already been deployed in hardware. Using this meta-data, the editor tool can connect directly to remote hardware devices running the code generated automatically and import the components implemented on that node. This way, the automatic C code generation tool contains data structures with information about the original model elements, including components, input and output signals and events, internal signals and Petri net nodes. This information may be queried by remote clients.

The HTTP server implements a set of HTTP requests corresponding to the procedure calls presented in table 2, grouped according to several request types.

Type	Procedure calls
User authentication	Login(), Logout(), RequestPriv(level)
Model metadata	GetModelName(), GetModelURL(), EnumerateComponents(), GetComponentInterface(comp), ListModeMetadata()
Read data	GetAttrValues(attr-list) SubscribeChanges(attr-list)
Write data	SetAttrValues(list,values), TriggerEvents(list)
Synchronization	GrabInputs(list), GrabComponent(comp-id), ReleaseInputs(list), ReleseComppnent(comp-id)
Trace & debug	ResetExecution(), StartExecution(), StopExecution(), ExecSteps(n), GetTraceMode(), DefineBreakpoints(list), GetBreakpoints(), GetActiveBreakpoint()
Force values	ForceValues(list), ThawForcedValues(list)
Dynamic reconfiguration	(Not yet implemented)

Table 2: Remote procedure calls

These remote procedure calls are invoked using standard HTTP requests and the results are encoded as JSON objects, as presented in the communication excerpt on listing 1.

On a distributed application, each node runs an HTTP server that may have multiple simultaneous clients. As a consequence, these clients may try to execute conflicting actions. While component outputs may be subscribed by multiple clients, the same is not valid for component inputs: at any time each input signal may only be driven by a single source. One exception are input events, that in some cases may be triggered by multiple clients without conflicts. For example, a component controlling a sensor might receive events from multiple clients to trigger sensor reading operations. To prevent conflicts, the «GrabInputs()» and «GrabComponent()» synchronization procedure calls are used to request exclusive access to specific inputs or all inputs of a component.

In order to optimize bandwidth and reduce latency, clients may subscribe notifications about value changes on specific attributes. For example, when an input of a local component is driven from the output of a remote component, the automatic code generator will subscribe change notifications from the remote server. As a result, the remote server keeps a persistent connection open, used to transmit a stream of JSON objects containing the new values of changed subscribed attributes and triggered events. Subscribed notifications are implemented using HTTP server side events [28], that are interpreted on the client side to update local copies of the remote values. In contrast, remote attributes driven by local components must be pushed to remote nodes using the «SetAttrValues()» and «TriggerEvents()» procedure calls.

When a connection drops, the client code automatically retries to establish the connection. However, while a connections is down, the receptor components that receive data from the remote node must be notified about the communication failure. To solve this problem, each component input may be assigned an «on-error» value, that is automatically attributed by the client side of the communication code. This way, when a connection is down, the component inputs driven by remote signals are automatically set with the «on-error» value and the component internal logic may act accordingly. Depending on each specific application, the «on -error» may correspond to a neutral value that does not affect the system evolution, but in other cases is corresponds to an active error value that demands an immediate response.

Due to space limitations, detailed information about the client and server implementation were not included in this text. However, information about protocol details can be found in [29] and complete copies of the source code may be obtained by opening existing online models on the IOPT-Flow tool framework and invoking the C code generator.

VI. Validation applications

In order to evaluate the proposed communication protocol and the associated tool framework, a set of applications was prepared. One of these applications corresponds to the simple CPS application presented on figure 1 involving three nodes running on a local network.

Another application, presented in figures 6, 7 and 8, implements to a multi-user game. Although this application does not correspond to the typical use case of an industrial system, it is important to notice that entertainment is one of the application fields frequently listed as a potential target for CPS systems. Multi-user games are frequently implemented as distributed applications that may involve computational nodes (game servers) and nodes containing physical devices as sensors and input devices (buttons, joysticks, motion sensors, acceleration and pressure sensors, plus feedback actuators, etc.). This simple validation application uses a computational component that executes the game engine (figure 7) and two components associated to physical devices that implement the user interface for two users. When the user interface runs on a personal computer, running inside a Web browser or as a standalone application, the input devices are the computer mouse or keyboard. However, when the code is deployed on other hardware, the user input may be performed using other hardware as physical push buttons.

Due to the interactive nature of games, possible software glitches or networking delays are immediately noticed by the users, providing an ideal test-bed to validate the entire tool framework. However, as the main goal of the application was the validation of the new concepts and not the game quality, a very simple «pong» game was selected. Figure 6 presents the top models of two applications corresponding to two players. Both models use the same components, the game engine and a user interface, whose internal models are available online at the IOPT-Flow framework page.

The game must be started by the first player, whose application runs both components locally, the game engine and the user interface. In contrast, the game engine component

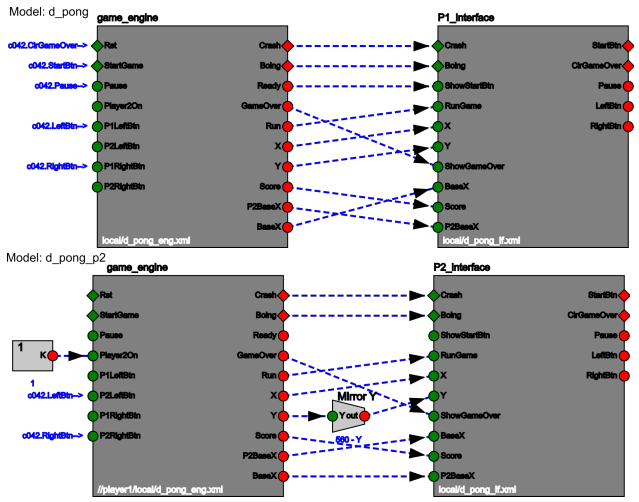

Fig. 6: Player1 (top) and Player2 (bottom) models

on the player 2 model is a remote component running on the player 1 application. This way, when the player2 application is launched, it connects to the first player application to access the game engine component shared by both users.

This applications was tested under different network configurations, ranging from a local network to a long distance Internet connection (500km). In the second case, although there were occasional communication pauses, the second user was still able to predict the ball position and continue playing.

VII. Conclusions

The communication protocol presented in this paper represents an advance regarding the previous version presented in [23]. Although the previous version already

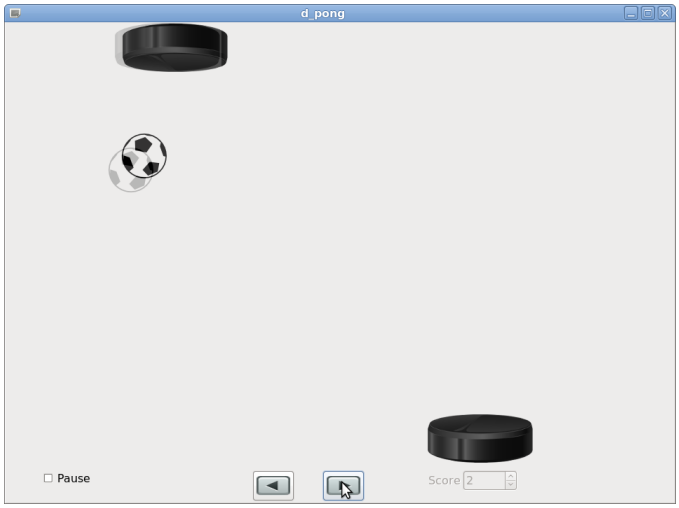

Fig. 8: Game user interface

offered remote debug and monitoring capabilities, it was designed to support a single connection to a centralized embedded controller. In contrast, the new protocol version was designed to build distributed applications, employing machine to machine communication between distributed nodes, and each node may receive multiple client connections. As a result, the new version of the protocol was added synchronization capabilities to avoid conflicts between applications trying to drive the same input signals and an improved user authentication subsystem with different privilege levels. As distributed applications usually involve components running on different locations, it is possible to build Web user interface applications to monitor sensors and information from multiple nodes and control actuators located on different nodes.

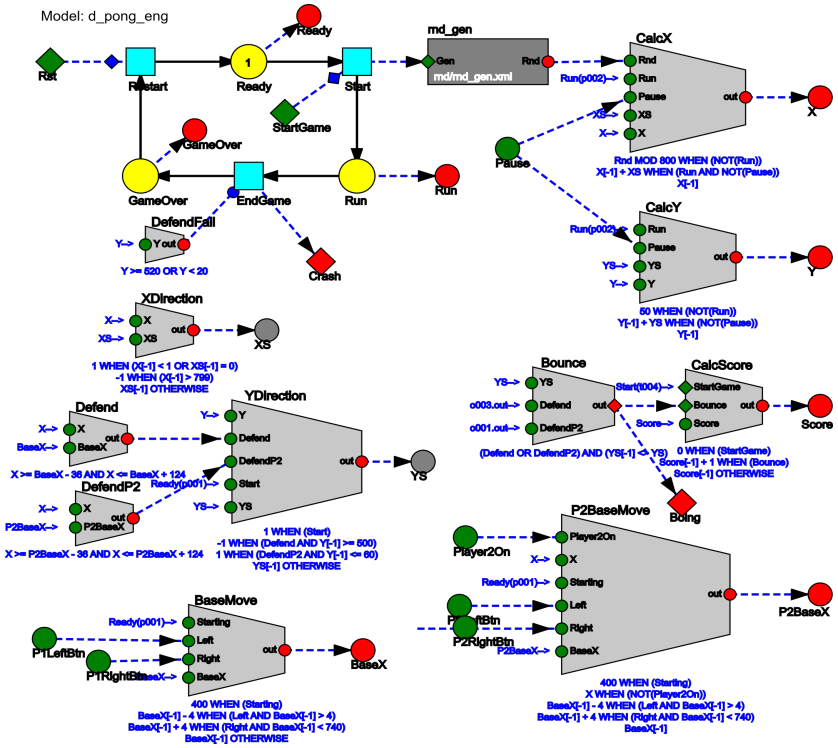

Fig. 7: DS-Pnet implementation of the game engine component

Communication performance results had already been presented in [23]. In the same way as the previous protocol, a typical connection between a client application and a remote node involves two communication channels, one to receive notifications about changes in subscribed values and other to drive remote inputs and send general commands. This way, the bandwidth and latency results obtained with the new version were similar to the data presented in [23] using two channels. However, as the previous protocol version did not supports the selection of subscribed values, it always transmitted data about entire models. In contrast, the new version of the protocol enables the selection of subscribed values and will offer higher performance when only a subset of the remote data is subscribed.

Using the proposed protocol and the IOPT-Flow tool framework, it is possible to create distributed applications using only graphical models, without needing to manually write any lines of code, as happened with the game example presented in this paper. All code was produced by automatic code generation tools, including the compilation of model meta-data and JSON/HTTP client and server code.

Planned future work include enhancements to the existing protocol, including HTTP transport data compression and TLS encription [30], support for centralized authentication services as Kerberos [31], LDAP [32] or RADIUS [33]. Interoperability with devices using other IoT/M2M protocols as CoAP[26], MQTT[25] and OPC/UA[27], may be achieved building proxy software that creates DS-Pnet representations of those devices and employ those protocols to communicate with these devices.

Acknowledgment

This work was partially financed by Portuguese Agency FCT – Fundação para a Ciência e Tecnologia, in the framework of project UID/EEA/00066/2013

References

[1] Baheti, R., & Gill, H. (2011). Cyber-physical systems. The impact of control technology, 12, 161-166.

[2] Gunes, V.; Peter, S.; Givargis, T.; Vahid, F.; A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems. KSII Transactions on Internet and Information Systems (TIIS), 8.

[3] Khaitan, S. K.; McCalley,J. D.; "Design Techniques and Applications of Cyberphysical Systems: A Survey," in IEEE Systems Journal, vol. 9, no. 2, pp. 350-365, June 2015, doi: 10.1109/JSYST.2014.2322503

[4] Monostori, L. (2014). Cyber-physical production systems: Roots, expectations and R&D challenges. Procedia CIRP, 17, 9-13.

[5] Pereira, F.; Gomes, L.; "Combining data-flows and petri nets for cyber-physical systems specification", Technological Innovation for Cyber-Physical Systems - 7th IFIP WG 5.5/SOCOLNET Advanced Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2016, Proceedings. Vol. 470 2016. p. 65-76 (IFIP Advances in Information and Communication Technology; Vol. 470)

[6] Reisig, W., "Petri nets: an introduction"; New York, USA: SpringerVerlag New York, 1985

[7] Gomes, L.; Barros, J.; Costa, A.; Nunes R., "The Input-Output Place-Transition Petri Net Class and Associated Tools", INDIN'2007 - 5th IEEE International Conference on Industrial Informatics, 23-26 July 2007, Vienna, Austria

[8] Lee, E.A.; Messerschmitt, D.G., "Synchronous data flow," in *Proceedings of the IEEE* , vol.75, no.9, pp.1235-1245, Sept. 1987 doi: 10.1109/PROC.1987.13876

[9] Tripakis, S., Bui, D., Geilen, M., Rodiers, B., & Lee, E. A. (2013). Compositionality in synchronous data flow: Modular code generation from hierarchical sdf graphs. ACM Transactions on Embedded Computing Systems (TECS), 12(3), 83.

[10] https://www.w3schools.com/js/default.asp [accessed 2018/4/15]

[11] IEEE Standard VHDL Language Reference Manual," in IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002) , vol., no., pp.c1-626, Jan. 26 2009, doi: 10.1109/IEEESTD.2009.4772740

[12] Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format.

[13] Pereira, F.; Gomes, L.; "The IOPT-Flow framework pairing Petri nets and data-flows for embedded controller development", IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, 2016, pp. 4832-4837. doi: 10.1109/IECON.2016.7794152

[14] Pereira, F.; Moutinho, F.; Gomes, L., "IOPT-Tools - Towards cloud design automation of digital controllers with Petri nets". ICMC'2014 International Conference on Mechatronics and Control. July 03-05 2014

[15] Pereira F.; Gomes, L.; "The IOPT-Flow Modeling Framework Applied to Power Electronics Controllers," in IEEE Transactions on Industrial Electronics, vol. 64, no. 3, pp. 2363-2372, March 2017; doi: 10.1109/TIE.2016.2620101

[16] Pereira, F.; Moutinho, F.; Gomes L.; Campos-Rebelo, R; "IOPT Petri net state space generation algorithm with maximal-step execution semantics", 9th IEEE International Conference on Industrial Informatics, INDIN 2011, Caparica, Lisbon

[17] Hanisch, H. M., & Lüder, A. (2000). A signal extension for Petri nets and its use in controller design. Fundamenta informaticae, 41(4), 415-431.

[18] Jensen, K.;"Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use" - Volume 1 Basic Concepts. Berlin. Germany.: SpringerVerlag., 1997.

[19] Kummer, O.; Wienberg, F.; Duvigneau, M.; Cabac, L.; "Renew – User Guide ", University of Hamburg, Department for Informatics, Theoretical Foundations Group, Release 2.2, August 28, 2009

[20] Hamez, A.; Hillah, L.; Kordon, F.; Linard, A.; Paviot-Adet, E.;Renault, X.; Thierry-Mieg, X.; "New features in CPN-AMI 3: focusing on the analysis of complex distributed systems," Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on, pp.273-275, 28-30 June 2006, doi: 10.1109/ACSD.2006.15

[21] Starke, P.H.; Hanisch, H.-M., "Analysis of signal/event nets," in Emerging Technologies and Factory Automation Proceedings, 1997. ETFA '97., 1997 6th International Conference on, vol., no., pp.253-257, 9-12 Sep 1997, doi: 10.1109/ETFA.1997.616278

[22] Davis II, J., Goel, M., Hylands, C., Kienhuis, B., Lee, E. A., Liu, J., ... & Smyth, N. (1999). Overview of the Ptolemy project (Vol. 99). ERL Technical Report UCB/ERL.

[23] Pereira, F., Melo, A., & Gomes, L. (2015, July). Remote operation of embedded controllers designed using IOPT Petri-nets. In Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on (pp. 572-579). IEEE.

[24] Botta, A., De Donato, W., Persico, V., & Pescapé, A. (2016). Integration of cloud computing and internet of things: a survey. Future Generation Computer Systems, 56, 684-700.

[25] Stanford-Clark, A., & Truong, H. L. (2013). Mqtt for sensor networks (mqtt-sn) protocol specification. International business machines (IBM) Corporation version, 1.

[26] Shelby, Z., Hartke, K., & Bormann, C. (2014). The constrained application protocol (CoAP).

[27] Van der Linden, D., Granzer, W., & Kastner, W. (2011, July). OPC Unified Architecture (OPC UA) new opportunities of system integration and information modelling in automation systems. In Industrial Informatics (INDIN), 2011 9th IEEE International Conference on (pp. 1-169). IEEE.

[28] https://www.w3schools.com/html/html5_serversentevents.asp

[29] Pereira, F.; "The DS-Pnet modeling formalismfor cyber-physical system development", 2017, https://run.unl.pt/handle/10362/27876

[30] Kerberos: The Network Authentication Protocol, https://web.mit.edu/kerberos/ (accessed Sep. 2017)

[31] Harrison, R.; Lightweight directory access protocol (LDAP): Authentication methods and security mechanisms, 2006

[32] Hill, J.; An analysis of the RADIUS authentication protocol. InfoGard Laboratories, 2001TLS