IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 May 2023, accepted 5 June 2023, date of publication 19 June 2023, date of current version 28 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3287198

RESEARCH ARTICLE

Integrated Development of Embedded Systems With Remote Graphical User Interfaces

CAROLINA LAGARTINHO-OLIVEIRA[1,2], (Graduate Student Member, IEEE), FERNANDO PEREIRA[2,3], (Member, IEEE), FILIPE MOUTINHO[1,2], ROGERIO CAMPOS-REBELO[1,2,4], AND LUIS GOMES[1,2], (Senior Member, IEEE)

[1]NOVA School of Science and Technology, NOVA University Lisbon, 2829-516 Caparica, Portugal
[2]Center of Technology and Systems, Institute for the Development of New Technologies (UNINOVA), NOVA University Lisbon, 2829-516 Caparica, Portugal
[3]Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, 1959-007 Lisbon, Portugal
[4]School of Technology and Management (STM), Polytechnic Institute of Beja, 7800-295 Beja, Portugal

Corresponding author: Carolina Lagartinho-Oliveira (ci.oliveira@campus.fct.unl.pt)

This work was supported in part by the Portuguese Agency "Fundação para a Ciência e a Tecnologia" (FCT) under Project UIDB/00066/2020 and Project UIDP/00066/2020, and in part by the Ph.D. Scholarship through national funds from the "Ministério da Ciência, Tecnologia e Ensino Superior (MCTES)" under Grant 2020.08462.BD.

ABSTRACT This paper presents a Graphical User Interface (GUI) Builder for embedded systems and Cyber-Physical Systems (CPSs). The hardware platforms used in CPSs may employ physical devices without graphical and user-input capabilities, or devices that may be placed on multiple remote locations, often with difficult physical access, that brings new challenges to GUI design. The proposed tool, integrated within the IOPT-Flow framework, enables a new approach to answer these challenges. This framework supports the development of embedded systems and distributed CPSs, offering a set of web-based tools for model edition, simulation, and implementation on physical devices. Systems are designed using graphical models combining Data-flows, Signals, and Petri nets, that permit the specification of remote communication channels just by drawing arcs. The new GUI Builder tool uses this infrastructure to automate the creation of graphical user interfaces for embedded systems and distributed CPSs, benefiting from a development with combined design, validation and automatic code generation methods. Finally, an application example of a power wheelchair controller integrating a GUI, is presented.

INDEX TERMS Data-flow, embedded systems, GUI builder, model-driven development, Petri net, wheelchairs.

I. INTRODUCTION

The emergence of low cost electronic devices with communication capabilities permitted the transition from the traditional embedded systems, that often operated in an isolated fashion without remote communication, into an era of Internet-of-Things and Cyber-Physical Systems (CPSs) that focus on communication and system distribution. Cyber-physical systems are frequently based on networks of distributed nodes, where individual nodes may provide computational or physical resources. The distributed nature of CPSs brought new challenges as a GUI may present information from multiple nodes located at different physical locations, operating asynchronously at different frequencies. In the same way, user interaction can affect multiple subsystems on different remote nodes. On CPS with multiple users, different GUIs may concurrently manipulate the same information. GUIs and models within a CPS are supposed to interact in ways that change with context, but their actions should not conflict with one another.

These GUIs monitor remote sensed values or control actuators, leaving the intensive data processing tasks to other nodes that may be deployed on the cloud, or on more specialized hardware, such as Graphics Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs). For example, the tool proposed in this paper can be applied to the development of GUIs to supervise a network of sensors and pumps, used to monitor and control the water flow of the

The associate editor coordinating the review of this manuscript and approving it for publication was Ton Duc Do.

rivers crossing a metropolitan area; to monitor city traffic in real-time, or even to monitor a fleet of wheelchairs, issuing alerts on low battery status and long periods of senior person inactivity.

Furthermore, traditional languages and tools do not offer combined design, analysis, and deployment methods, useful to support the development of distributed CPSs, leading to the separated implementation of the distributed components. This paper focuses on this research gap, and proposes an approach that supports an integrated view of complete CPS systems, presenting distributed components side-by-side connected with arcs.

The new GUI Builder was integrated into the IOPT-Flow tool framework that offers a set of web-based tools for the design of CPSs, allowing the edition and simulation of individual components as well as the entire systems [1]. For example, the IOPT-Flow web-based simulation environment directly supports the graphical objects designed with the new GUI Builder, and presents them interacting with the other model's components. This way, the developer can verify and validate the entire CPS systems (with GUIs) before implementing them on real devices. The final implementation is assisted by automatic code generators that produce code to deploy CPS models on each node, and also the code to establish their communication, using a JSON/HTTP protocol.

The models are specified using the DS-Pnet formalism [2], which combines the characteristics of two proven formalisms: Petri nets [3] and data-flows [4]. As data processing plays an important role in CPSs, data-flows and signals can provide a graphical formalism to specify the dependencies between input and output signals, applying mathematical transformations and performing data processing operations. On the other side, Petri nets [5] are used to create state-based models that define the evolution of system state. Concretely, Petri net models: (1) enable the modeling and visualization of system behavior with parallelism, concurrency, synchronization, and resource sharing; (2) have precise syntax and execution semantics that support the verification and validation of CPSs, as well as their implementation, using design automation tools; (3) support bottom-up and top-down modeling strategies, associated with the composition and decomposition of models. Additionally, to support the creation of distributed systems, the DS-Pnet formalism offers the concept of components – which enclosure/encapsulate models – that may be placed locally or in remote locations.

The main proposed contribution of this paper is a new GUI Builder tool that supports an integrated approach for distributed CPS GUI design, extending the IOPT-Flow tool framework. It permits the design of graphical user interfaces just by dragging and dropping widgets (graphical objects) inside GUI window frames, and automatically creating the respective DS-Pnet components. The GUI code is generated automatically, contributing to minimize development time and errors.

An application example is presented to validate the implementation of a CPS that provides an alternative way to control a power wheelchair by integrating a GUI. The system was implemented on a Raspberry Pi, and it was possible to interact with the system using the GUI locally and remotely.

This paper is organized with the following structure: begins with the related work and an introduction to the DS-Pnets and the IOPT-Flow tools; next, the proposed GUI Builder is presented, with a description of the meta-model of the GUI that can be created with it, and the algorithm employed to generate the DS-Pnet GUI components; and finally a validation example and conclusions.

II. RELATED WORK

CPSs are leading researchers to propose new approaches and tools for GUI development, providing new possibilities in how GUIs can be used, not necessarily restricted to the primary application - as typically occurs in environments like Eclipse WindowBuilder [6]. This feature enables CPSs to support different behavioral modalities and allows GUIs within CPSs to be used in different contexts. Currently available tools include works to adapt existing interfaces [7], create interfaces for existing services [8], methods to support rapid prototyping, portability, and usability, among others. Many of these works have considered the extension of markup languages, the use of platform-independent vocabularies and toolkits, and even the adoption of model-driven development (MDD) approaches [9].

There are many ways to specify GUIs. The Interaction Flow Modeling Language (IFML) [10], a platform-independent modeling language from the Object Management Group (OMG), was designed to express the content, user interaction, and control behavior for software application front-ends. eXtensible Markup Languages (XML) [11], such as User Interface Markup Language (UIML) [12] or USer Interface eXtensible Markup Language (UsiXML) [13], address abstract models, allowing platform independence, and the reuse of elements previously described, in new GUIs. In [14], UsiXML is used to specify abstract GUI models that are transformed into code for specific platforms, using eXtensible Stylesheet Language for Transformation (XSLT). In another approach, Unified Modeling Language (UML) diagrams can be used to describe interfaces, as described in [15].

Examples of GUI development environments include TERESA [16], which supports the design of interfaces accessible through various device types in a web-based environment; GrafiXML [17], a graphical editor for UsiXML; MobiGUITAR [18], a GUI testing framework; VAQUITA [19], a tool to convert HyperText Markup Language (HTML) pages into other representation formats, based on eXtensible Interface Markup Language (XIML); MD2 [20], a model-driven framework were an application is described in Domain-Specific Language (DSL), and transformed into

code source for a specific platform; MIMIC [21], that relies on the M4L language that uses state machines to model interfaces; and [22] presents a GUI development workflow that begins with the creation of a platform-independent model that is later transformed to target different specific platforms.

With a focus on the implementation of embedded systems and CPSs, Petri nets are a modeling formalism suitable for their specification [3]. Today, different classes of Petri nets are frequently used, ranging from low-level Petri nets as the Input-Output Place-Transition (IOPT) Petri net class [5], [23], to high-level Petri nets as Coloured Petri nets (CPN) [24], which simplify the description of systems that carry complex information associated with data processing. BRITNeY [25] is a Petri-net-based tool that allows the animation of formal models. As it is integrated with CPN tools [26], CPN models can trigger specific actions in the resulting GUI. Animator and Synoptic [27] are two tools that support the automatic generation of GUIs for IOPT Petri net models. A GUI designed using Animator contains a set of rules that define the relation between the GUI elements and the embedded system model. The Synoptic tool controls the execution of this model and updates the GUI.

The purpose of the GUI Builder presented in this paper is to provide tools for the design and generation of GUIs for CPSs modeled with the IOPT-Flow framework. With a web-based user interface, the GUI Builder uses the HTML document object model for capturing the attributes and properties of the GUI widgets and produces an XML file containing a DS-Pnet model that describes the GUI elements and a set of attributes used to interface with the other CPS subsystems. This way, the resulting model contains a set of DS-Pnet components that implement the designed GUI elements. The IOPT-Flow framework is used to connect the inputs and outputs of the GUI elements to the remaining CPS subsystems, allowing the construction of complex CPS systems. Next, the automatic code generation tools are used to implement the controllers and their GUIs on specific target platforms. Since the new tool is integrated within a Petri net-based framework, the resulting systems can be verified and validated using the available Petri net model-checking tools.

When compared to Animator and Synoptic tools, the proposed GUI Builder allows the design of GUIs independently of any control model for which they are created, enabling the creation of GUI libraries to be reused, and is not constrained to a limited rules editor, supporting the creation of more complex graphical interfaces. Contrary to most traditional tools that just provide user interfaces for single embedded devices, the new generation GUI Builders target distributed CPSs where the GUI objects can interact directly with physical devices (sensors, actuators, controllers) located on remote nodes, and must support multiple GUIs for different users interacting with the same CPS. In this category, we can find tools like Node-Red for IoT devices [28], n8n [29], Outsystems [30], and the new GUI Builder here proposed. However, only GUI Builder supports model-checking.

The case study in this paper demonstrates that a GUI can be used to monitor and control the speed and seat position of a power wheelchair. As it was designed to validate the new tool, it does not implement all the functionalities available on existing commercial products. For example, the MyLiNX app [31], a power wheelchair diagnostics app for Invacare wheelchairs using a LiNX control system [32], provides access to the basic system and diagnostic information from users' wheelchairs and enables them to share this information with a technician. Another app named My Permobil [33] offers enhanced wheelchair performance and usage data. It offers easy access to information about the battery charge, travel distance, and power seat functions.

III. DS-PNETS AND IOPT-FLOW TOOL FRAMEWORK

The GUI Builder was added, in this work, to the IOPT-Flow development framework [1], used to enable the rapid development of embedded systems and distributed CPSs. This framework contains a set of web-based tools that support all phases of system development, including:

1) Web based model design and edition;
2) Online simulation and model-checking tools to permit the early detection of design mistakes, before deployment on the embedded devices;
3) Automatic code generation tools to implement model semantics on different hardware targets (C, Java-script, VHDL) including a communication layer to manage the communication between distributed components;
4) Graphical remote debug and monitor, to enable problem detection and resolution after system deployment;
5) A library of previously created components to accelerate system development, including timers, communication protocols, memory components and file I/O.

Models developed using the IOPT-Flow framework are based on the DS-Pnet (data-flows, signals, and Petri nets) formalism [2]. DS-Pnet models may be divided into components that encapsulate different subsystems, enabling the usage of top-down or bottom-up development approaches and the creation of component libraries. The external interface of a DS-Pnet component is composed of input and output signals and events. Multiple DS-Pnet components can be deployed on different hardware nodes to implement an entire CPS system. As the interfaces produced by the new GUI Builder are composed of DS-Pnet models, the resulting GUI appears as a component that may be seamlessly inserted into the models of the entire CPSs. So, independently if a component implements a GUI or another subsystem, input and output signals and events allow communication between the parts.

CPS systems developed using DS-Pnet models use IOPT Petri nets to specify system behavior, which evolves in response to external input signals. The relationship between input and output signals and Petri net nodes is specified using data-flow operations and arcs, that may apply mathematical transformations to perform signal processing and conditioning. For instance, data-flow operations can be applied to filter

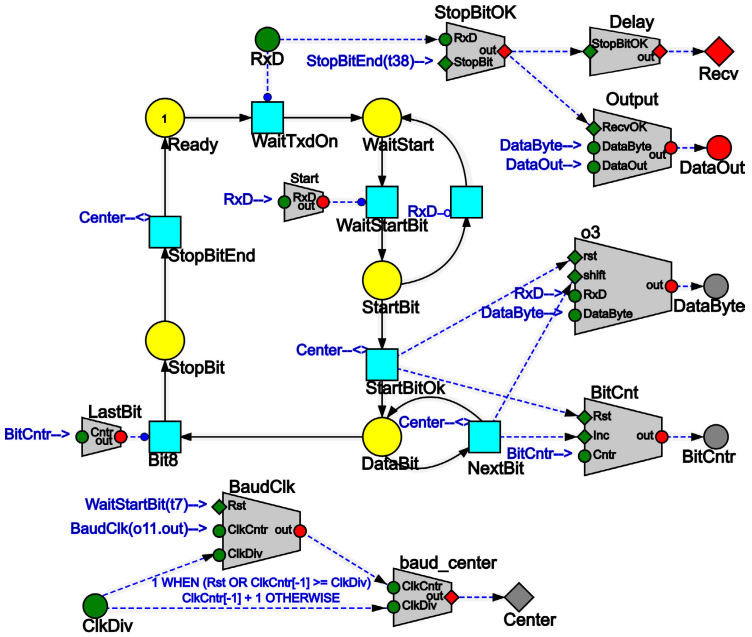


FIGURE 1. Example of a DS-Pnet model: An UART receiver, with input, internal and output signals/events as green, gray or red circles/diamonds, Petri net places as yellow circles, transitions as cyan rectangles, data operations as gray quadrilaterals, and arcs as arrows or reference textual; the name of each element is in bold and the conditions or guards are presented with text below operations and beside the transitions.

low-level sensor data and define guard conditions and input events used to regulate the system evolution. Other data-flow operations may also be used to calculate output signals based on the internal system state and input signals.

The Petri net part of a DS-Pnet model inherits the characteristics from the parent IOPT-net class [5]. Execution is performed in discrete steps, employing a maximal step execution semantics, meaning that all enabled transitions are forced to immediately fire on the next execution step. Conflicts between transitions, that compete for tokens from the same places, are solved by assigning priorities to transitions. The evolution of the system's state is controlled using transition input events and guard conditions.

Fig. 1 exhibits a model created in the IOPT-Flow editor. This model implements a functional UART receiver that has been synthesized (and tested) on FPGAs using the VHDL automatic code generator. The external interface of this model is composed by two input signals, RxD and ClkDiv (displayed as green circles); and two outputs, a DataOut output signal (red circle) and a Recv output event (red diamond). The evolution of the system state is controlled by a Petri net, where Petri net places are presented as yellow circles and transitions as cyan rectangles. All mathematical operations are performed using data-flow operations, presented as gray quadrilaterals. In this example, the data-flow operations are used to shift the bits received from the RxD input and calculate DataOut (output bytes) or to create a counter to divide the system clock and compute the desired baud rate. Fig. 2 presents the DS-Pnet component of the described model,

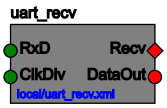


FIGURE 2. Example of a DS-Pnet component: An UART Receiver.

in which the model is encapsulated in a gray rectangle, presenting an interface with the accessible inputs and outputs.

Due to the distributed nature of CPSs, the GUI interface must deal with several challenges. To cope with these challenges, the DS-Pnet components produced by the GUI Builder take advantage of the IOPT-Flow infrastructure, including the automatic generation of the execution semantics and communication code. This means that a GUI component can be transparently connected to multiple distributes nodes, to present information and control remote devices. For instance, it is possible to display the value of remote sensors in real-time just by connecting an arc between the sensor output to GUI inputs used to display graphical wave-forms; in the same way, it is possible to connect an arc from a GUI Button output to an input on a remote node used to start/stop an electrical motor.

On multi-user systems with more than one GUI, the system must prevent conflicts between different concurrent user actions. The underlying communication subsystem that connects different DS-Pnet components does not allow more than one client to simultaneously control the same input. This way, when such a situation is required, the designer

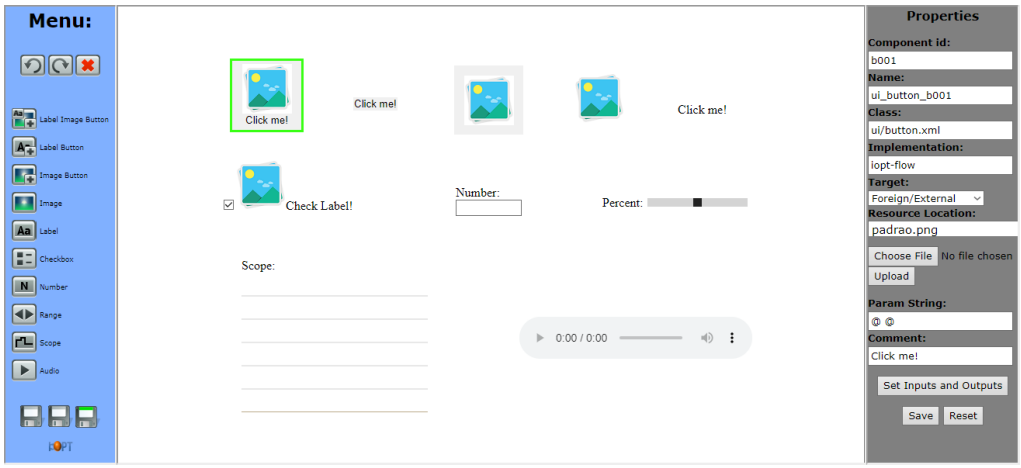


FIGURE 3. GUI builder interface.

must manually model the DS-Pnet glue logic, for instance, by defining priorities, or simply by using the and/or logic operators. When conflicts occur, the user interface logic may also be programmed to hide or inhibit particular GUI elements.

The manual resolution of conflicts constitutes a limitation of the proposed approach. However, the designer may employ several design patterns to solve conflicts. For example, a component offering a boolean input to control a physical resource cannot be simultaneously operated by multiple users. However, a design pattern that offers two separate input events (set and reset), does not suffer from this problem, as the system just memorizes the state of the last action. If both event occur in the same execution step, the system will prioritize one of them.

In addition, the aspect and behavior of the user interface may dynamically change according to the context and the state of the remote physical devices. To deal with this, the user interface components also contain inputs used to control the GUI widgets parameters. For example, certain GUI buttons may only be visible when the respective hardware devices are ready; the graphical position of certain GUI widgets may change in real-time in order to reflect the position of their physical counterparts; the sensitive status of a GUI widget can also be dynamically controlled, according to the evolution of system state; the user interface may also switch from multiple views whenever the context changes.

IV. THE GUI BUILDER

The proposed GUI Builder enables the design of graphical user interfaces, providing widgets that can be positioned in the design canvas by dragging and dropping them to the desired location. A form with properties is used to customize these widgets' attributes, according to the system requirements. The interface of GUI Builder is presented in Fig. 3. The GUI can be tested at any time by automatically converting the canvas content into a DS-Pnet model containing components corresponding to each GUI widget, which may be executed by the IOPT-Flow debugger.

A. GUI BUILDER AND THE IOPT-FLOW FRAMEWORK

Fig. 4 gives an overview of the proposed workflow approach. The developer can edit a GUI in the GUI Builder and use it in the IOPT-Flow framework. As a result, the new tool automatically creates a component representing the designed GUI model, which is subsequently added to the DS-Pnet model of the entire CPS system, where the GUI widgets may be connected to the rest of the subsystem components; then, the existing IOPT-Flow tools can be applied to the resulting model. For example, the simulator can be used to perform the system's execution on a web interface and detect semantic errors; after successful simulation, the automatic code generation tool produces implementation code to deploy the system into specific software and hardware devices; the remote debugger supports the debug of models running on local or remote hardware. The resulting GUI can be used locally or remotely via Web.

The interaction between the GUI Builder and the IOPT-Flow framework is presented in Fig. 5. The GUI Builder is launched through the IOPT-Flow Editor, which is the main interface of the IOPT-Flow Framework. During the GUI edition, the new tool represents the designed interface on the screen using a DOM model containing an HTML representation of all GUI elements, which are updated in real-time in response to the user interaction. This representation was chosen, as DOM objects can be directly visualized on the web browser without needing to be transformed. They can also be extended dynamically with new attributes and methods, in order to support the requirements and specific functionalities of the project.

Although the HTML documents are stored in the IOPT-Flow framework, these HTML documents are just internal representations that are not used by the remaining IOPT-Flow tools. Instead, the GUI elements are converted

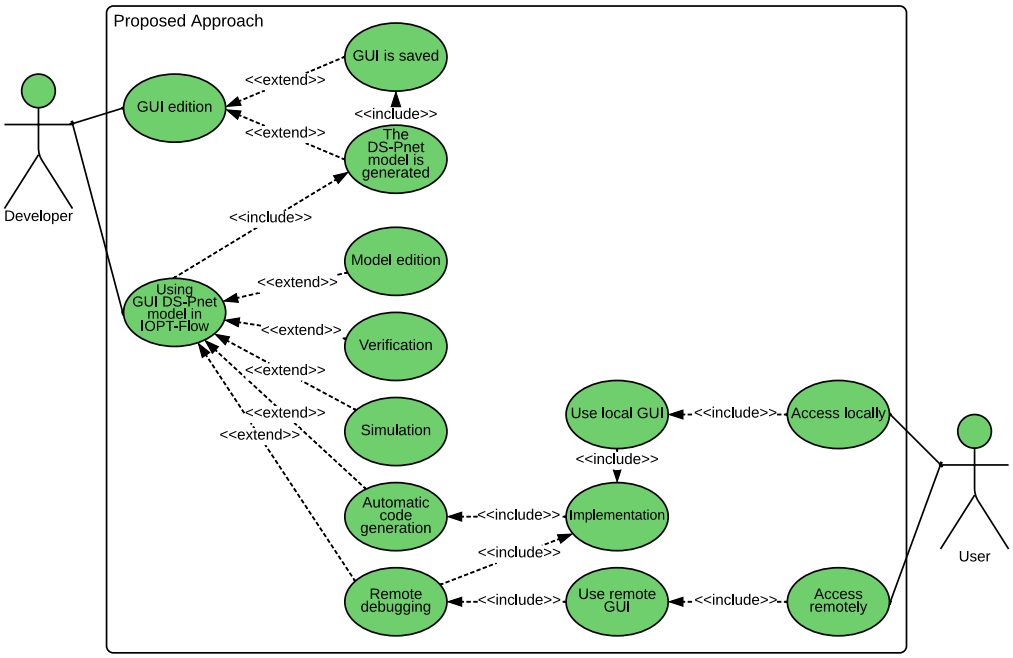


FIGURE 4. A use case diagram showing how GUIs are developed and used using the proposed approach.

into DS-Pnet models and components and stored as XML files. As Fig. 5 also presents, instead of HTML, the other tools of the IOPT-Flow framework employ a graphical representation of the models, based on SVG graphics that are produced by applying XSLT transformations to the XML GUI files. When the remaining tools operate over models containing GUI elements, they must implement the behavior of the GUI widgets. For instance, the web simulator and debugger present a browser HTML window with the GUI interface that may be connected to local simulator components or to remote physical devices. In the same way, the automatic code generator tools also produce GUI code respecting the properties of the GUI model. For example, the C code generator implements the GUI interfaces using the GTK+ library [34].

B. ENVIRONMENT

The proposed tool allows rapid prototyping of GUIs that can be integrated with DS-Pnet-based controllers without the need for manual programming. The GUI Builder presents an interface identical to the IOPT-Flow, with a toolbox; a form with the widget properties; and a drawing area.

As the internal representation of the GUI elements is stored as an HTML DOM document. The DOM is structured as a set of nodes and objects with properties, methods, and events. All user actions performed in the GUI Builder are interpreted by JavaScript functions running on the browser, producing changes to the DOM elements. JavaScript functions are invoked by events that allow, for example, to create, modify, remove and recover the content of the DOM document. Operations used to manage data storage are implemented using PHP code running on the server.

A designer's first action when starting a new GUI is to add new widgets to the drawing area by choosing them from the toolbox. The toolbox offers several tools dedicated to the edition and automatic generation of GUIs, highlighting the following set of widgets.

- Button - presents a button with an image or/and a label;
- Image - presents a static image;
- Label - presents a static text message;
- Checkbox - tick box that can have a label;
- Number - numeric inputs;
- Range - presents a scroll-bar/scale;
- Scope - graphical scopes to display wave-forms in real-time;
- Audio - plays a sound sample.

The toolbox also offers options to support file management operations: to create, update and open project HTML files; and to save or update the resources used in those projects. Another button is used to create or update the resulting GUI DS-Pnet model and the respective component symbol to be inserted in CPS models.

The GUI Builder presents an area dedicated to the edition of the widget's properties. When a widget is dragged or selected, a form appears displaying the properties of the widget. A second form presents the input and output parameters of the widgets: as the GUI widgets are going to be connected to other CPS elements, for instance, sensors and actuators, the widgets' properties must be defined to ensure data type compatibility with these elements.

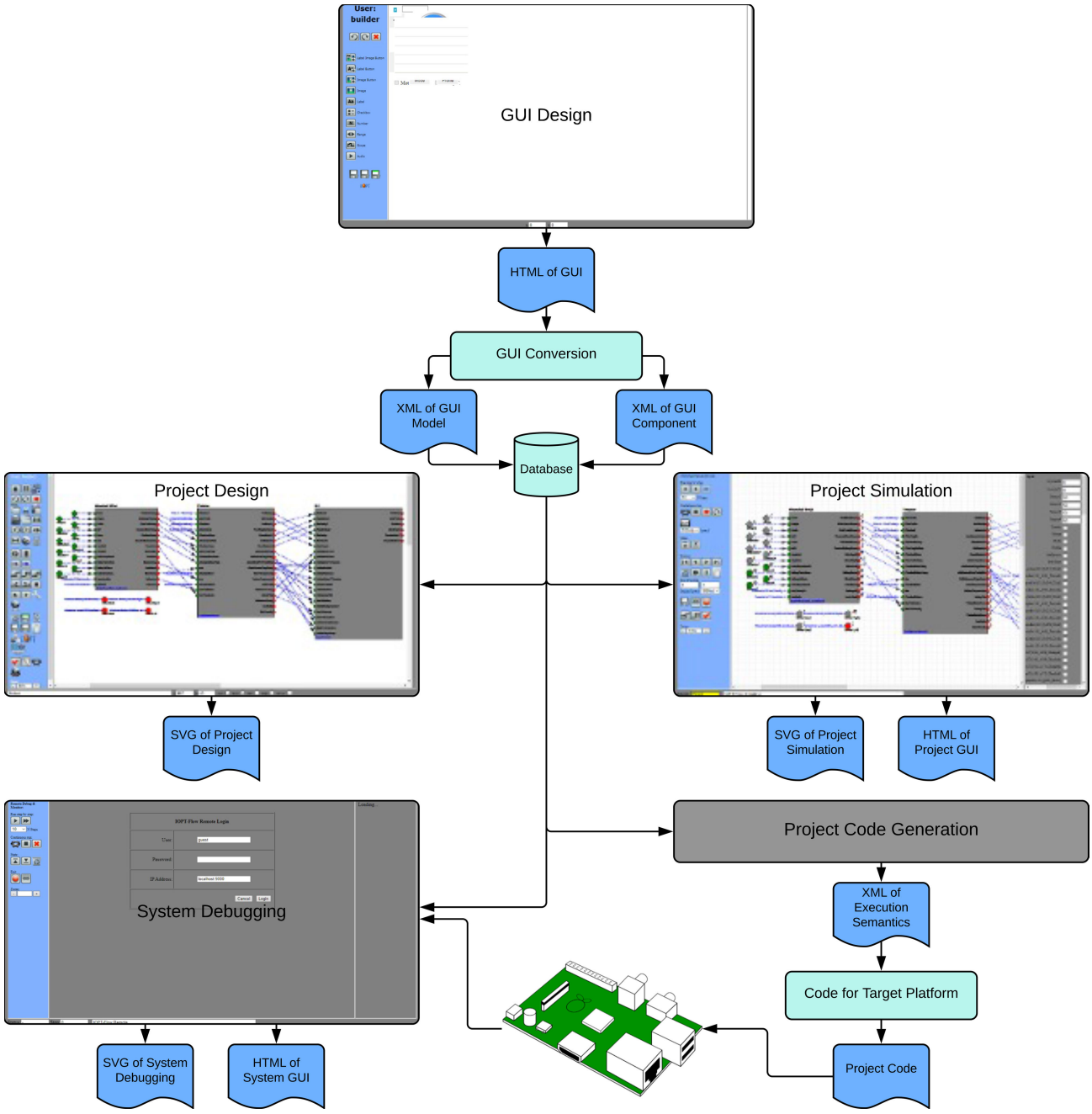


FIGURE 5. Architecture that relates GUI builder with IOPT-Flow framework.

C. GUI META-MODEL

Figures 6 and 7 present a part of GUI meta-model. Common widgets attributes are the identifier, name, target device selection, parameter string, and comment. Some widgets require a source file, such as ≪wav≫ audio files or ≪png≫ images.

Each GUI element/widget has several inputs to define the values to be displayed and control the graphical position and size, visibility, and sensitivity (e.g. a frozen visible widget that is not responsive). Some widgets have events to update the values displayed or to trigger actions, for instance, to play sound samples. In the opposite direction, the widgets offer outputs to report user actions, for instance, when a button is pressed or a value has been changed, or when a sound sample is playing. For example, Figure 7 presents the button's structure. Each input and output may be associated with a source and target element. In this case, a source element may either be a constant or a signal; and a target element can be either a signal or an event.

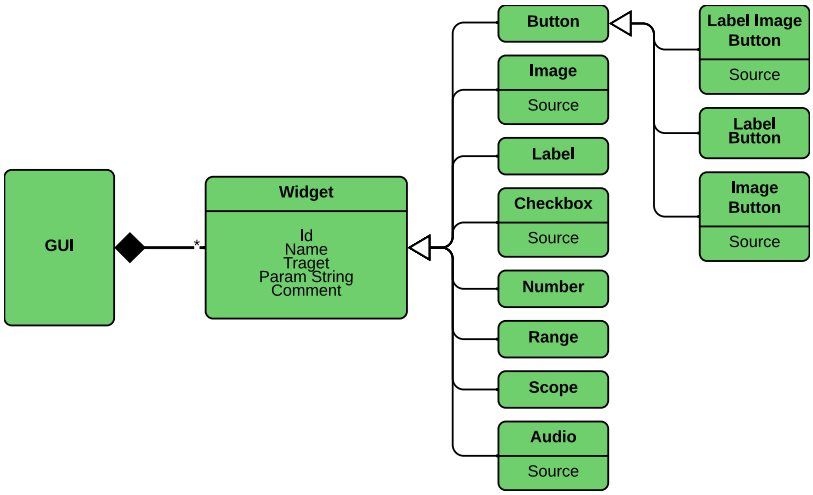


FIGURE 6. The core structure of the GUI meta-model.

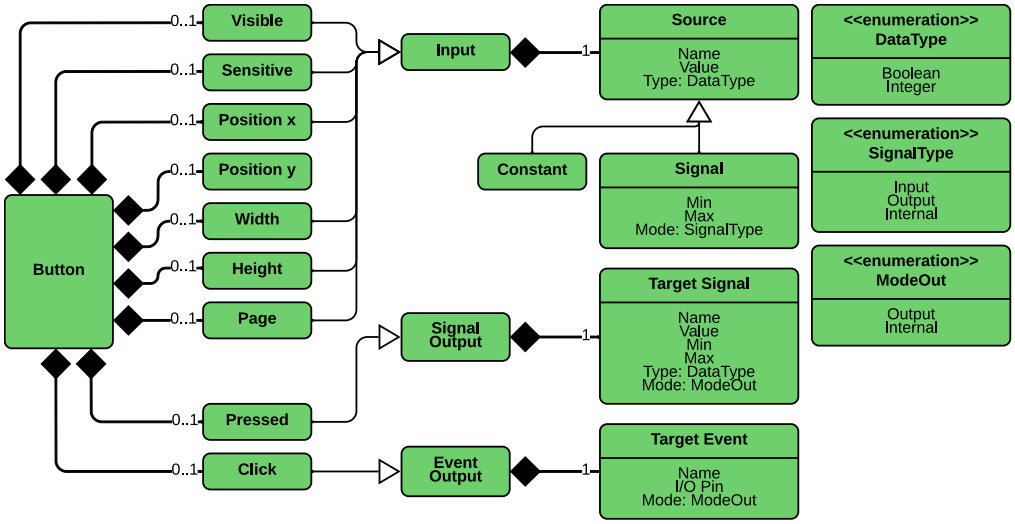


FIGURE 7. The button structure of the GUI meta-model.

D. GENERATION OF GUI DS-PNET MODELS AND COMPONENTS

The output produced by the GUI Builder is composed of a DS-Pnet model containing the GUI elements, and a component symbol, representing this model. The Algorithm 1, used to create these models, generates the following items:

- Widgets components;
- Input, internal and output signals;
- Constants;
- Input and output events;
- Arcs.

Observing this list, in addition to the widgets' generation, the algorithm also produces a list of inputs and outputs that define the internal interface of the GUI itself.

The resulting models are stored on the server as XML files.

It is important to note that although XML files are created from the information contained in the HTML file of the designed GUI, the project can be continuously edited in the GUI Builder without interfering with the already generated DS-Pnet model and component, which are only generated when the user requests. The opposite is also true since the resulting DS-Pnet model can be edited in the IOPT-Flow Editor without changing the GUI design in the GUI Builder. As the internal representation of the GUI is independent of the IOPT-Flow framework, it may be possible to convert the interfaces designed for other formalisms and development languages.

V. APPLICATION EXAMPLE

This section presents a graphical user interface developed in the GUI Builder that can be applied to control and display the parameters of a power wheelchair, similar to the interfaces often found on commercial power wheelchairs controlled with a joystick. The interface and all the mod-

Algorithm 1 GUI Model & Component Generation

```
data ← HTML DOM model
repeat
    for each widget node in data do
        create widget XML
        for each defined io in widget node do
            create io XML
            create read arc XML{Connect the io and widget XML}
        end for
    end for
until IOPT-Flow model of the GUI is created
if GUI project exist then
    update model in the IOPT-Flow files folder
    update component in the IOPT-Flow library folder
else
    create new project
    save model in the IOPT-Flow files folder
    create IOPT-Flow component{Based on the model}
    save component in the IOPT-Flow library folder
end if
```

els and components that illustrate the example are currently available in GUI Builder (http://gres.uninova.pt/iopt-flow/inter_clo/inter_clo.html) and IOPT-Flow environments (http://gres.uninova.pt/iopt-flow/), accessible through the user *builder* (with read-only permissions), with the same password.

This particular example implements a simple GUI to be used as an alternative way to control the speed and seat of a power wheelchair, as well as to visualize and monitor the parameters of its motors and battery.

To allow the validation of GUI, it was used in a DS-Pnet model connected to other two components: one modeling the power wheelchair (''Wheelchair Model''); and the other controlling the GUI logic and the wheelchair's values (''Presenter''). The interaction between the three components is shown in Fig. 11.

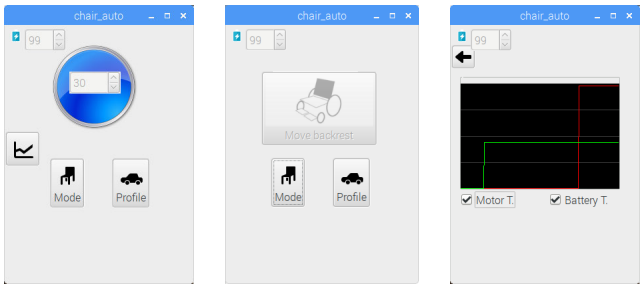
The IOPT-Flow design automation tools were used to test and deploy the project, including the simulator tool, remote debugger, and automatic code generator. The example was simulated, and its C code was executed on a Raspberry Pi 3 device that interacted remotely with the IOPT-Flow remote debugger.

A. GUI

The example application allows users to change the operating mode and profile of a wheelchair, monitor the velocity, read the state of the battery charge, and display waveforms of measured values. Fig. 8 presents some of the interface pages created. The interface works as follows:

- The homepage displays the wheelchair's speed and three buttons: ''Mode'', ''Profile'' and ''Graphs'';
- The ''Mode'' button allows the user to switch between the mode to drive, and raise or tilt the wheelchair;
- The ''Profile'' button allows switching between two speed ranges, to increase or decrease the velocity;
- The ''Graphs'' button allows viewing the waveforms of specific measurements;
- The page used to display waveforms has checkboxes to select the graphs to be presented, namely motor and battery temperature measurements;
- All pages display the battery charge status in the upper left corner.

After creating the interface and customizing all elements, the GUI was converted into a DS-Pnet model. Observing Fig. 8 it is possible to check if the aspect of the GUI elements respects the intended behavior, and confirm if they correspond to the DS-Pnet generated components.



(a) Homepage (b) Mode 1 (c) Measurements

FIGURE 8. Interface designed for a power wheelchair.

Inspecting the resulting DS-Pnet model (available online), it contains 14 widget components, and the widget parameters have been customized with constant values or assigned to external inputs and outputs. These inputs and outputs are internally connected to the widget components through the automatic drawing of arcs. The component on the right side of Fig. 11 represents the entire GUI and the external inputs and outputs used to connect the GUI to the remaining components (''Presenter'' and ''Wheelchair Model'').

In this application example, the GUI component has 23 inputs and 6 outputs. Seven inputs events (''SetBattery'', ''SetVelocity'', ''SetMotorT'', ''SetBatteryT'', ''NewSample'', ''ResetMotorT'' and ''ResetBatteryT'') are used to trigger the update of the values presented in the interface, namely the value of the battery charge, the speed of the wheelchair, and the values used to present graphs in the scope. Input signals whose name starts with ''Visible'' are used to manage the appearance of a widget in the interface; and the ''NewValBattery'', ''NewValVelocity'', ''MotorTemperature'' and ''BatteryTemperature'' input signals refer to the values presented in the interface when the events mentioned above occur. In the opposite direction, the GUI component has 4 output events associated with button clicks, that trigger actions in the other components; and 2 output signals to select the graphs to be presented in the scope.

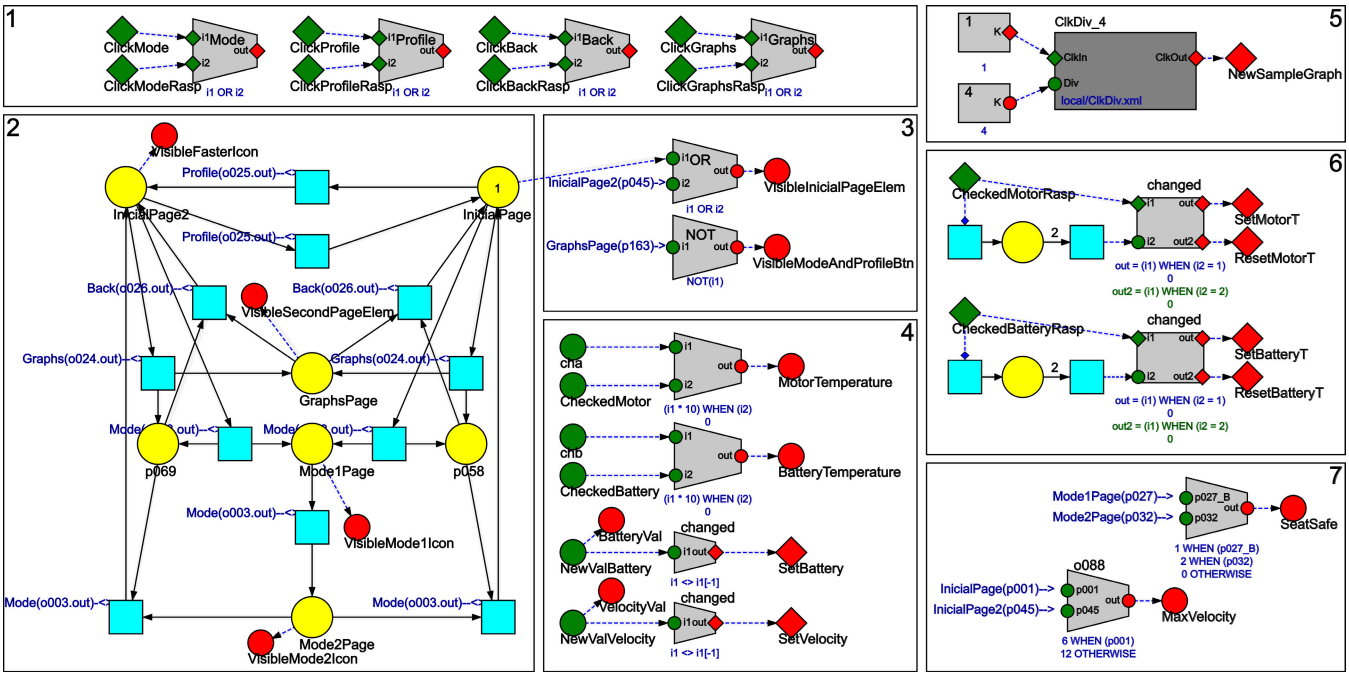


FIGURE 9. Model of "Presenter".

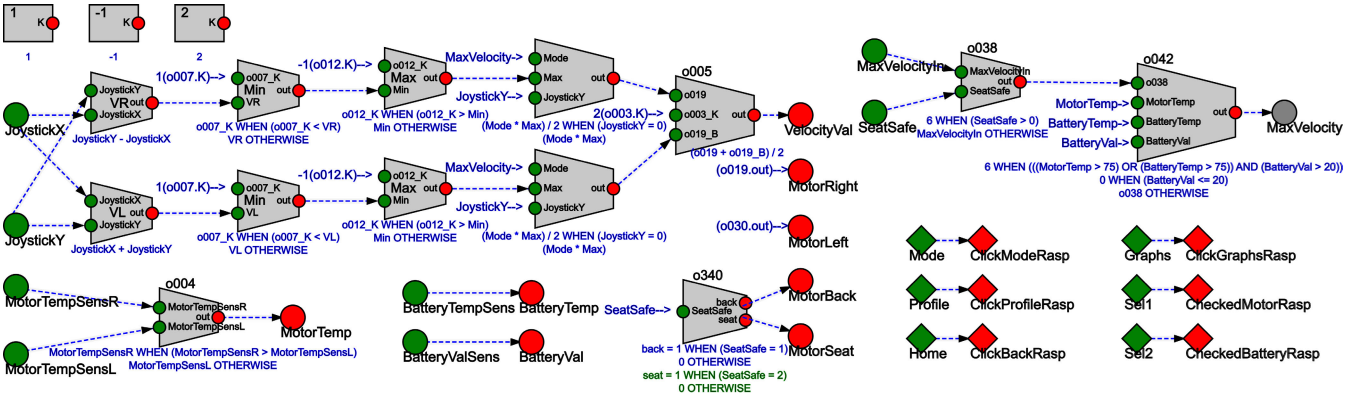


FIGURE 10. Model of "wheelchair model".

B. PRESENTER

Fig. 9 presents the model encapsulated by the component "Presenter"; 7 areas were highlighted in the model:

- In 1, the gray components correspond to "or" operations between the respective input events, which correspond to the detection of a click on a GUI button;
- In 2 there is a Petri net with 5 places corresponding to the pages of the GUI; the other elements allow to identify the user commands;
- The Petri net output signals are used to define which elements of the interface are visible at a given moment; two of them are affected by the conditions presented in 3;
- In 4, the values presented in the waveforms are selected according to two checkboxes, and events are generated to trigger the update of GUI numeric widgets;
- In 5, the "NewSampleGraph" event is created, forcing the scroll of the scope waveforms with the addition of new samples;
- The status of the checkboxes is saved in 6, so that consecutive clicks toggle the status of a checkbox;
- In 7, two components are used to output the maximum allowable velocity value for the wheelchair, depending on "Profile" chosen, and a flag to inform when the user is using some seat function, to raise or tilt the wheelchair.

C. WHEELCHAIR MODEL

Next, the "Wheelchair Model" is shown in Fig. 10. The "Wheelchair Model" maps the position of the x and y axes of the joystick to obtain the velocity value of each motor

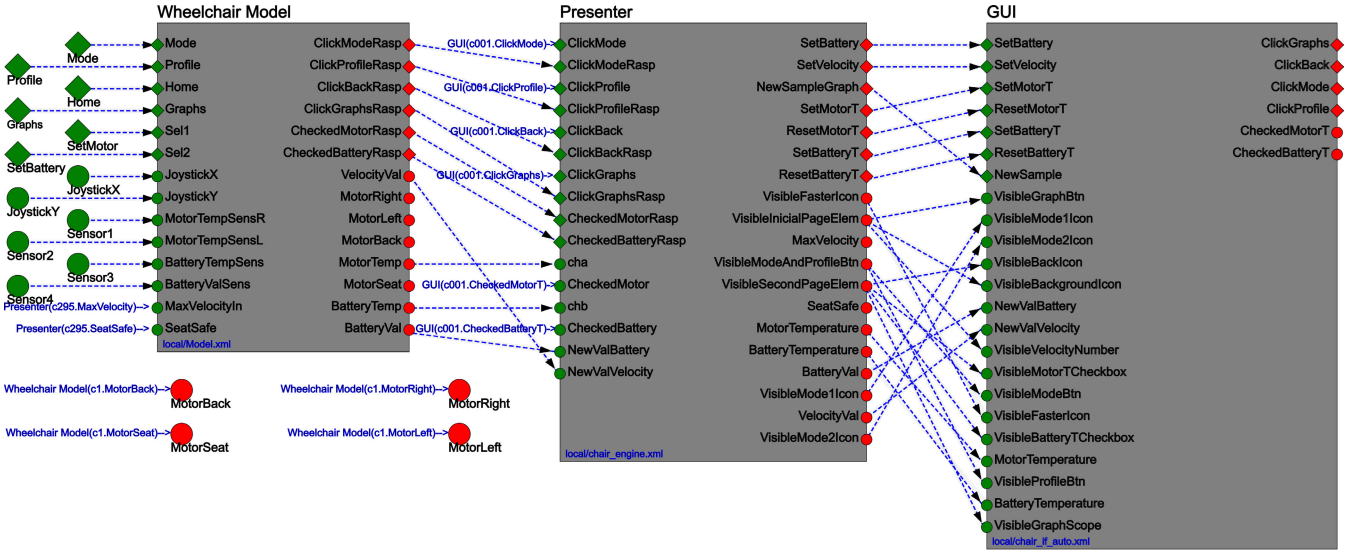


FIGURE 11. DS-Pnet model of an alternative control device for a power wheelchair with integrated GUI.

which composes the power wheelchair (''MotorLeft'' and ''MotorRight'') and sends the longitudinal velocity value to the ''Presenter''.

The longitudinal velocity is conditioned by the input value of the battery level, the temperature of the battery and each of the motors, and the value of the maximum allowable velocity and the state of the seat received from ''Presenter''. The seat state value is also used to actuate the ''MotorBack'' and ''MotorSeat'' motors, used to tilt or raise the wheelchair.

The ''Wheelchair Model'' also defines the data to be displayed in the GUI, by sending information to the ''Presenter'' to update values of the temperature and battery charge, the highest temperature among the motors, and whether the user wants to change the wheelchair profile, mode, see graphics, etc. In alternative to the GUI, the user can also use the physical buttons available on the wheelchair (shown at the bottom right side of Fig. 10).

On the left side of Fig. 11, the ''Wheelchair Model'' component reads the status of the joystick status, physical buttons, and motor and battery sensors. The ''Presenter'' component in the center, acts upon the ''Wheelchair Model'' limiting and retrieving values from it and controlling the values displayed on the GUI. On the right, the ''GUI'' component, automatically generated by the GUI Builder, displays data and routes additional user commands to the ''Presenter''.

D. RESULTS VALIDATION

To validate the correct operation of the interface, two types of analysis were carried out. First, the simulator of the IOPT-Flow tool framework was used to check the evolution of the model over time (token-player), forcing values to the input signals and events connected to the ''Wheelchair Model'' component. During simulations, the simulator tool presents a Web version of the generated GUI, allowing direct interaction with it.

After that, a second type of validation through physical experimentation was performed using the automatic C code generation tool and the remote debugger/monitor application, enabling the execution and visualization of the system state in quasi-real-time. The automatic C code generator was used to deploy the model of Fig. 11 and its executable application, built and compiled using a Raspberry Pi 3 board; the Raspberry Pi also established communication with the remote debugger to support the remote debug and monitoring. The entire build process did not involve manually writing code; the code ran directly on the Raspberry Pi, displaying the interface of Fig. 8 locally, using the Linux operating system, and remotely in the remote debugger.

In addition, a simple circuit was implemented on a breadboard, presenting a set of physical buttons and variable resistors connected to the platform, so that the user sends commands directly to the ''Wheelchair Model''. The buttons placed on the breadboard were used to allow the occurrence of events related to the GUI, namely those that represent the click on ''Mode'', ''Profile'', ''Back'', ''Graphs'', and the checkboxes related to the temperature graphs; the variable resistors were used to simulate the x and y axes of a joystick, as well as values for battery levels and temperatures. Thus, the user was able to interact with the system using 3 different ways: through the generated GUI, changing values on the remote debugger tool, or using the sensors and buttons connected to the GPIO ports of Raspberry Pi. During the execution of the controller, it was possible to confirm the correctness of the GUI.

VI. CONCLUSION

This paper presents the GUI Builder, a Web-based tool that allows the edition, and automatic generation of graphical user interfaces. The GUI Builder is integrated into the IOPT-Flow tool framework, available online at

http://gres.uninova.pt/iopt-flow/ and free to use, supporting the rapid development of embedded and distributed cyber-physical systems with GUIs.

GUIs for distributed systems can be rapidly developed, just by dragging and dropping widgets on a canvas, and adjusting their properties according to application requirements. As all the code is generated automatically, the occurrence of coding errors is minimized, contributing also to rapid prototyping and deployment of GUIs. The resulting GUI components can be used to locally or remotely monitor, control, or animate a behavioral model of a CPS system developed in the IOPT-Flow Editor. The communication between the GUI and the other CPS subsystems is defined just by drawing arcs linking the inputs and outputs of the GUI components and the system's components, and does not require any manual coding.

The GUI Builder supports an approach where the GUI is developed separately from the main CPS model. As a result, a developer designing a GUI for a CPS does not need to understand the internal details of the remaining CPS subsystems, it just requires basic knowledge of them and the parameters required for user interaction. This feature makes GUI Builder projects independent of any behavioral model, which means that it can be easily integrated with other development tools and modeling formalisms. This approach also simplifies the support for multi-user CPS that require different GUIs for each user, developed as separate projects.

To validate the GUI Builder, a scenario was presented to control, view, and monitor the parameters of a power wheelchair. The validation, of the GUI design and generation, started with an analysis of the automatically generated XML files. Then, the GUI component was integrated with the components, ''Wheelchair Model'' and ''Presenter'' which model the entire system operation, and together they were implemented and tested using the IOPT-Flow tool framework. To perform the remote simulation, the remote debugger tool, the automatic C code generator, and a Raspberry Pi 3 were used. The results were consistent with those of the local simulation, making it possible to observe the correct functioning of the GUI. With the validations carried out, it was also possible to verify the correct implementation of the GUI Builder.

Although the presented application example is not very complex, the same approach can be used to implement GUI for other types of applications and industries. For example, to build GUIs and dashboards for industrial production lines, home automation systems, railways and transportation systems, smart grids, and many other applications.

The design applications can take advantage of the design automation tools offered by the IOPT-Flow tool framework. This framework supports all phases of system development, including edition, simulation and model-checking, implementation through automatic code generation, and remote debugger. Safety-critical applications may benefit from the model-checking tools to verify compliance with safety regulations. It is important to notice that the code generated automatically supports the communications between remote components, which enables the deployment of GUIs for distributed systems in a transparent way.

As future work, we consider the improvement of the GUI Builder, to improve user experience with additional types of widgets, support different hardware devices, and allow the generation of GUIs in other modeling formalisms. In order to enhance the capabilities of the GUI builder, more complex use cases must be employed, interfacing with components located on multiple distributed hardware devices.

REFERENCES

[1] F. Pereira and L. Gomes, ''The IOPT-flow modeling framework applied to power electronics controllers,'' *IEEE Trans. Ind. Electron.*, vol. 64, no. 3, pp. 2363–2372, Mar. 2017, doi: 10.1109/TIE.2016.2620101.

[2] F. Pereira and L. Gomes, ''Combining data-flows and Petri nets for cyber-physical systems specification,'' in *Technological Innovation for Cyber-Physical Systems*, L. M. Camarinha-Matos, A. J. Falcão, N. Vafaei, and S. Najdi, Eds. Cham, Switzerland: Springer, 2016, pp. 65–76, doi: 10.1007/978-3-319-31165-4_7.

[3] C. Girault and R. Valk, *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Berlin, Germany: Springer, 2003, doi: 10.1007/978-3-662-05324-9.

[4] E. Yourdon, ''Structured programming and structured design as art forms,'' in *Proc. Nat. Comput. Conf. Expo.*, May 1975, p. 277, doi: 10.1145/1499949.1499997.

[5] L. Gomes, F. Moutinho, F. Pereira, J. Ribeiro, A. Costa, and J. Barros, ''Extending input–output place-transition Petri nets for distributed controller systems development,'' in *Proc. Int. Conf. Mechtron. Control (ICMC)*, Jul. 2014, pp. 1099–1104, doi: 10.1109/ICMC.2014.7231723.

[6] Eclipse Foundation. *WindowBuilder*. Accessed: Sep. 2020. [Online]. Available: https://www.eclipse.org/windowbuilder/

[7] M. H. Ramli, A. H. Jantan, A. Kamaruddin, and R. H. Abdullah, ''The adaptive model driven approach for enhancing usability of user interface design: A review process,'' in *Proc. 5th Int. ACM In-Cooperation HCI UX Conf.*, Apr. 2019, pp. 65–69, doi: 10.1145/3328243.3328252.

[8] H. Bünder, ''A model-driven approach for graphical user interface modernization reusing legacy services,'' Univ. Münster, Eur. Res. Center Inf. Syst., Münster, Germany, ERCIS Working Papers 30, 2019. [Online]. Available: http://hdl.handle.net/10419/203466

[9] J. Ruiz, E. Serral, and M. Snoeck, ''Evaluating user interface generation approaches: Model-based versus model-driven development,'' *Softw., Syst. Model.*, vol. 18, no. 4, pp. 2753–2776, Aug. 2019, doi: 10.1007/s10270-018-0698-x.

[10] S. Roubi, M. Erramdani, and S. Mbarki, ''A model driven approach to generate graphical user interfaces for Rich Internet Applications using Interaction Flow Modeling Language,'' in *Proc. 15th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Dec. 2015, pp. 272–276, doi: 10.1109/ISDA.2015.7489237.

[11] S. D. Rathod, ''Automatic code generation with business logic by capturing attributes from user interface via XML,'' in *Proc. Int. Conf. Electr., Electron., Optim. Techn. (ICEEOT)*, Mar. 2016, pp. 1480–1484, doi: 10.1109/ICEEOT.2016.7754929.

[12] J. Helms, R. Schaefer, K. Luyten, J. Vermeulen, M. Abrams, A. Coyette, and J. Vanderdonckt, ''Human-centered engineering of interactive systems with the user interface markup language,'' in *Human-Centered Software Engineering: Software Engineering Models, Patterns and Architectures for HCI*, A. Seffah, J. Vanderdonckt, and M. C. Desmarais, Eds. London, U.K.: Springer, 2009, pp. 139–171, doi: 10.1007/978-1-84800-907-3_7.

[13] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero, ''USIXML: A language supporting multi-path development of user interfaces,'' in *Engineering Human Computer Interaction and Interactive Systems*, R. Bastide, P. Palanque, and J. Roth, Eds. Berlin, Germany: Springer, 2005, pp. 200–220, doi: 10.1007/11431879_12.

[14] F. J. Martinez-ruiz, J. Munoz Arteaga, J. Vanderdonckt, J. M. Gonzalez-calleros, and R. Mendoza, ''A first draft of a model-driven method for designing graphical user interfaces of Rich Internet Applications,'' in *Proc. 4th Latin Amer. Web Congr.*, Oct. 2006, pp. 32–38, doi: 10.1109/LA-WEB.2006.1.

[15] S. Roubi, M. Erramdani, and S. Mbarki, "Model driven architecture as an approach for modeling and generating graphical user interface," in *Proc. Medit. Conf. Inf. Commun. Technol.*, A. El Oualkadi, F. Choubani, and A. El Moussati, Eds. Cham, Switzerland: Springer, Apr. 2016, pp. 651–656, doi: 10.1007/978-3-319-30298-0_72.

[16] S. Berti, F. Correani, G. Mori, F. Paternò, and C. Santoro, "TERESA: A transformation-based environment for designing and developing multi-device interfaces," in *Extended Abstracts on Human Factors in Computing Systems*. Vienna, Austria: Association for Computing Machinery, Apr. 2004, pp. 793–794, doi: 10.1145/985921.985939.

[17] B. Michotte and J. Vanderdonckt, "GrafiXML, a multi-target user interface builder based on UsiXML," in *Proc. 4th Int. Conf. Auton. Auto. Syst. (ICAS)*, Mar. 2008, pp. 15–22, doi: 10.1109/ICAS.2008.29.

[18] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "MobiGUITAR: Automated model-based testing of mobile apps," *IEEE Softw.*, vol. 32, no. 5, pp. 53–59, Sep./Oct. 2015, doi: 10.1109/MS.2014.55.

[19] L. Bouillon and J. Vanderdonckt, "Retargeting web pages to other computing platforms with VAQUITA," in *Proc. 9th Working Conf. Reverse Eng.*, Nov. 2002, pp. 339–348, doi: 10.1109/WCRE.2002.1173091.

[20] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with MD^2," in *Proc. 28th Annu. ACM Symp. Appl. Comput. (SAC)*. Coimbra, Portugal: Association for Computing Machinery, Mar. 2013, pp. 526–533, doi: 10.1145/2480362.2480464.

[21] N. Elouali, X. Le Pallec, J. Rouillard, and J.-C. Tarby, "A model-based approach for engineering multimodal mobile interactions," in *Proc. 12th Int. Conf. Adv. Mobile Comput. Multimedia (MoMM)*. Kaohsiung, Taiwan: Association for Computing Machinery, Dec. 2014, pp. 52–61, doi: 10.1145/2684103.2684112.

[22] A. Schuler and B. Franz, "Rule-based generation of mobile user interfaces," in *Proc. 10th Int. Conf. Inf. Technol., New Generat.*, Apr. 2013, pp. 267–272, doi: 10.1109/ITNG.2013.43.

[23] F. Pereira, F. Moutinho, A. Costa, J.-P. Barros, R. Campos-Rebelo, and L. Gomes, "IOPT-tools—From executable models to automatic code generation for embedded controllers development," in *Proc. 43rd Int. Conf. Appl. Theory Petri Nets Concurrency (PETRI NETS)*, L. Bernardinello and L. Petrucci, Eds. Bergen, Norway: Springer, Jun. 2022, pp. 127–138, doi: 10.1007/978-3-031-06653-5_7.

[24] K. Jensen and L. M. Kristensen, "Colored Petri nets: A graphical language for formal modeling and validation of concurrent systems," *Commun. ACM*, vol. 58, no. 6, pp. 61–70, May 2015, doi: 10.1145/2663340.

[25] M. Westergaard and K. B. Lassen, "The BRITNeY suite animation tool," in *Petri Nets Other Models Concurrency—ICATPN*, S. Donatelli and P. S. Thiagarajan, Eds. Berlin, Germany: Springer, 2006, pp. 431–440, doi: 10.1007/11767589_26.

[26] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, nos. 3–4, pp. 213–254, Jun. 2007, doi: 10.1007/s10009-007-0038-x.

[27] L. Gomes and J. Lourenco, "Rapid prototyping of graphical user interfaces for Petri-net-based controllers," *IEEE Trans. Ind. Electron.*, vol. 57, no. 5, pp. 1806–1813, May 2010, doi: 10.1109/TIE.2009.2031188.

[28] G. R. Kanagachidambaresan, "Node-red programming and page GUI builder for industry 4.0 dashboard design," in *Role of Single Board Computers (SBCs) in Rapid IoT Prototyping*. Cham, Switzerland: Springer, May 2021, pp. 121–140 doi: 10.1007/978-3-030-72957-8_6.

[29] n8n. *A Powerful Workflow Automation Tool*. Accessed: May 2023. [Online]. Available: https://n8n.io/

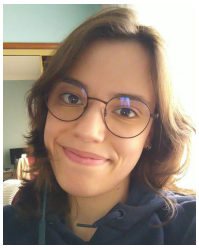
[30] R. Martins, F. Caldeira, F. Sá, M. Abbasi, and P. Martins, "An overview on how to develop a low-code application using OutSystems," in *Proc. Int. Conf. Smart Technol. Comput., Electr. Electron. (ICSTCEE)*, Oct. 2020, pp. 395–401, doi: 10.1109/ICSTCEE49637.2020.9277404.

[31] Invacare. *MyLiNX*. Accessed: Mar. 2021. [Online]. Available: https://play.google.com/store/apps/details?id=com.dynamiccontrols.mylinx&hl=pt&gl=US

[32] Invacare Corporation. *Invacare LiNX Control System*. Accessed: Dec. 2022. [Online]. Available: https://www.invacare.co.uk/power-wheelchairs-mobility-scootersalber/power-chair-controls/invacare-linx-control-system

[33] Permobil. *MyPermobil*. Accessed: Mar. 2021. [Online]. Available: https://play.google.com/store/apps/details?id=com.permobil.sae.dockme&hl=pt_PT&gl=US

[34] GTK Team. *Create Apps That Users Just Love*. Accessed: Dec. 2022. [Online]. Available: https://www.gtk.org



CAROLINA LAGARTINHO-OLIVEIRA (Graduate Student Member, IEEE) received the M.Sc. degree in control systems, digital systems, and robotics from the Faculdade de Ciências e Tecnologia, NOVA University Lisbon, Portugal, in 2018, where she is currently pursuing the Ph.D. degree in computational and perceptual systems. Her research interests include the digital control of systems, Petri nets, embedded systems, parallel computing, and digital twin.

FERNANDO PEREIRA (Member, IEEE) received the degree in electrical engineering from the Technical University of Lisbon, in 1992, and the Ph.D. degree from NOVA University Lisbon, Portugal, in 2017. He is currently a Professor with the Polytechnic Institute of Lisbon, Lisbon, where he teaches computer programming. Over the past years, he has been collaborating with the industry in the fields of industrial automation, embedded systems, CAD/CAM, and computer numeric control. In addition to these topics, his research interests include the Linux operating systems, Petri nets, electronic design automation, and cyber-physical systems.

FILIPE MOUTINHO received the Ph.D. degree in electrical and computer engineering (digital and perceptional systems) from the NOVA School of Science and Technology (FCT NOVA), Portugal. He was a Software Engineer with Newhotel Software, Lisbon. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, FCT NOVA, and a Researcher with the Center of Technology and Systems (CTS), UNINOVA Institute, FCT NOVA. His main research interests include digital embedded systems, model-based development, and reconfigurable computing platforms.

ROGERIO CAMPOS-REBELO received the M.Sc. degree in electrotech engineering and the Ph.D. degree in electrical and computer engineering from Universidade Nova de Lisboa, Portugal, in 2010 and 2016, respectively. He is currently a Researcher with the NOVA School of Science and Technology and an invited Assistant Professor with the Polytechnic Institute of Beja, Portugal. He is also a member of the UNINOVA-CTS, Caparica, Portugal. His main research interests include formal methods, like Petri nets and other concurrence models, and the Internet of Things, applied to cyber-physical systems and Industry 4.0.

LUIS GOMES (Senior Member, IEEE) received the degree in electrotech engineering from the Technical University of Lisbon, Portugal, in 1981, and the Ph.D. degree in digital systems from NOVA University Lisbon, Portugal, in 1997. He was made an Honorary Professor with the Transilvania University of Brasov, Brasov, Romania, in 2007, and Óbuda University, Budapest, Hungary, in 2014. His main research interests include the usage of Petri nets and other models of concurrency, applied to reconfigurable and embedded systems co-design, and cyber-physical systems. He received the IEEE Industrial Electronics Society Anthony J. Hornfeck Service Award, in 2016.