Combining Low-Code Programming and SDL-Based Modeling with Snap*!* in the Industry 4.0 Context

Xavier Pi i Palomés, Pere Tuset-Peiró
Computer Science, Multimedia and Telecommunications Department
Universitat Oberta de Catalunya (UOC)
Barcelona, Spain
Email: {xpi, peretuset}@uoc.edu

Pau Fonseca i Casas
Statistics and Operation Research Department
Universitat Politècnica de Catalunya
BarcelonaTech (UPC)
Barcelona, Spain
Email: pau@fib.upc.edu

***Abstract*—One of the main challenges to implement Industry 4.0 technologies within the industrial fabric is the lack of suitable concrete models and tools that demonstrate the potential benefits of embracing the digital transformation process. To overcome this challenge, over the past years, various Industry 4.0 automation and robotics providers have presented solutions based on visual block programming languages, which follow an emerging low-code approach to reduce the entry barriers of digital technologies. However, block-based low-code tools typically lack the formality introduced by specification languages, limiting their ability to model the industrial processes formally. Taking this into account, in this article, we present the combination of specification languages and visual block programming languages to enable industrial users to test and/or build their own Digital Twin models at a suitable abstraction level and with low entry barriers. In particular, we combine SDL and Snap*!* to create SDL4Snap*!*, an open-source and web-based tool that facilitates the implementation and validation of Digital Twin prototypes. Overall, the resulting tool has the potential to reduce the entry barrier to Digital Twins in small and medium enterprises, which are part of the late majority and laggard groups regarding the adoption of digital technologies in the context of Industry 4.0.**

1. Introduction

The 4th Industrial Revolution, also known as Industry 4.0, is characterized by the digital transformation of the industrial fabric [1]. It is formalized through two main reference architectures: the Reference Architecture Model Industry 4.0 (RAMI 4.0), standardized as IEC/PAS 63088 [2], and the Industrial Internet Reference Architecture (IIRA) [3]. Both standards are based on the notion of Cyber-physical Systems (CPS), which rise from the coupling of the physical and digital worlds [4]. Physical systems can integrate electronic embedded systems, or simulators can integrate physical systems, becoming emulators [5].

Embracing simulation, data analytics and other engineering disciplines, the Digital Twin (DT) is regarded as the most important artifact of Industry 4.0 [6]. The term was first introduced in 2002 by Michael Grieves [7] and was later used by NASA as a Product Life-cycle Management (PLM) tool [8]. A Digital Twin is defined as a virtual reproduction of a system based on simulations, either with real-time or historical data, that allows representing, understanding, and predicting scenarios of the past, present, and future, with verified and validated models, and synchronized at a specified frequency and fidelity with the system [9].

According to the European Union Commission [10], the main limitation for the adoption of Industry 4.0 technologies in small and medium-sized enterprises currently lies in the lack of specific and concrete models and tools. In that regard, De Leeuw [11] has identified the existence of a minimum threshold of digital maturity in resources, systems, organization, and culture, required for a successful industrial digital transformation process. In the same direction, we use the term "suitable digital mindset for Industry 4.0" to denote the transversal model-based and computational thinking skills required to understand and apply the fundamental concepts, including Industry 4.0 architectures. A model-based low-code approach involving Digital Twins is a minimal core of this mindset. In that regard, Datta [12] explains that democratization of the *ad hoc* and *en masse* configuration of Digital Twins by non-experts will no longer limit the use and application of Digital Twins in the hands of experts alone, and leaders must proactively support to pursue collaborative initiatives.

In that direction, the low-code movement has recently emerged, aiming to remove barriers to use, configure, and build software for everybody, enabling a general digital cultural literacy and facilitating the challenge of interdisciplinary work. Given its potential, low-code is being promoted as a key infrastructure in the digital transformation landscape [13], and there is an increasing interest in low-code in the Industry 4.0 scope [14]. For example, Bosch-Rexroth introduced the "ctrlX developR" for automation systems, whereas ABB introduced the "ABB Wizard easy programming" for robotic applications. However, block-based low-code programming languages currently lack support for modeling, limiting their applicability to building and using Digital Twins due to a lower user empowerment.

Considering the importance of lowering the entry barrier to Digital Twins as a mechanism to foster the adoption of

978-1-6654-2484-4/21/$31.00 ©2021 IEEE
DOI 10.1109/MODELS-C53483.2021.00120

Industry 4.0 in small and medium enterprises, this work focuses on extending a visual block-based low-code programming language with modeling capabilities. To implement our tool, we have selected SDL and Snap*!* for two main reasons. On the one hand, SDL is a well-regarded standard graphical modeling language widely used to specify communications and simulation models that people from different disciplines can use. On the other hand, Snap*!* is an open-source and web-based visual block-based programming language that is widely used. For example, academia members such as Eckard Modrow [15], use Snap*!* in Computer Science courses because it covers many aspects of computer science from algoritmia, object/agent orientation, simulation, sound, and image processing to artificial intelligence.

Hence, the contributions of the article are the following. First, it introduces SDL4Snap*!*, a tool that facilitates the direct translation from a set of standard SDL blocks to Snap*!*. Second, it shows how to build a minimal implementation example, namely PingPong, from an SDL specification. Third, through a set of early experiments, it anticipates that combining model-based reasoning with low-code tools can help lowering the entry barrier to Digital Twins and fostering the adoption of Industry 4.0 technologies within the industrial fabric.

The remainder of the article is organized as follows. Section 2 presents the related work. Section 3 introduces the tools that have been used to create the SDL4Snap*!* tool. Section 4 gives an implementation example, namely PingPong, using the SDL4Snap*!* extension. Section 5 presents the results of a preliminary qualitative evaluation using the SDL4Snap*!* tool. Finally, Section 6 concludes the article.

2. Related Work

This section introduces the main concepts that are related to the topic of the paper: Industry 4.0 Architectures, Digital Twin and Low Code Programming.

2.1. Industry 4.0 Architectures

As stated earlier, Cyber-Physical Systems are a crucial building block of Industry 4.0. Cyber-Physical Systems are formally characterized by the reference architectures RAMI 4.0 [16] and IIRA [3]. RAMI has its origins in the OT (Operations Technologies) world and is standardized as IEC/PAS 63088 [2]. In contrast, IIRA has its roots in the IT (Information Technologies) world and is defined by the IIC (Industrial Internet Consortium). Hence, while RAMI focuses on the assets that compose the system, IIRA focuses on the network that interconnects the system. In 2018, a joint workgroup started to work on the harmonization of both architectures [17].

An interesting aspect of RAMI for CPS is that it represents assets through the notion of an Administration Shell, which relies on the metaphor that every artifact (real or virtual) can be covered with a "digital bell", giving rise to the Industry 4.0 Component (I4.0 Component). Therefore, an asset can be seen as an entity that exposes digital interface such as an API (Application Programming Interface) that allows it to be remotely monitored and controlled, partially or fully. In addition, RAMI specifies that the I4.0 Components must have the characteristics of nestability and encapsulability properties. The former means that every I4.0 Component can consist of other I4.0 Components which can be logically nested. The latter means that each I4.0 Component should be able to establish all necessary connections within an I4.0 System, and that they must be able to retain its core functionality even if the external network is disrupted [2]. Hence, a CPS can be defined as a composition of I4.0 Components that are connected through a network.

Regarding connectivity, both RAMI 4.0 and IIRA advocate for the servitization of assets based on supporting SOA (Service Oriented Architectures). RAMI 4.0 specifies that I4.0 Components must be SOA compliant [18], and according to Erl, [19] [20], this includes both client-server and event-driven architectures, such as publish-subscribe (PubSub). On the one hand, client-server architectures are based on the request-respond mechanism, which can be synchronous and asynchronous, as shown in Figure 1. In contrast, the PubSub architecture is based on the call-back mechanism and can be regarded as a generalization of client-server architectures. That is, in the PubSub architecture communications are not limited to one-to-one, but one-to-many communications are also supported.

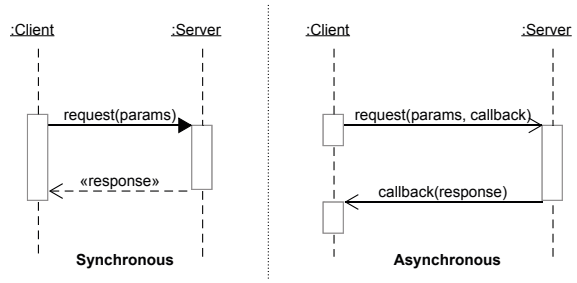


Figure 1. Client-server architectures

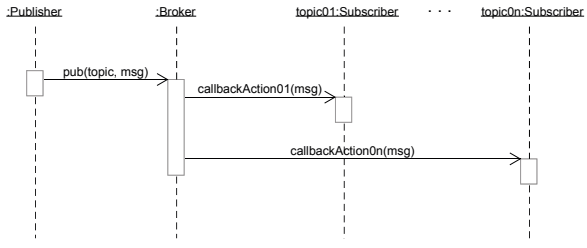


Figure 2. PubSub architectures

As shown in Figure 2, in a PubSub architecture the publisher agent sends a message to an intermediate agent

named broker, indicating a channel (named as topic). Upon receiving a message, the broker forwards it to all subscribed agents. Hence, PubSub architectures have an asynchronous nature. However, it is possible to implement an equivalent synchronous request-response behavior if there is only one subscriber for the request and the publisher is also the only subscriber of the response [19].

Finally, it is important to mention that RAMI 4.0 is technologically agnostic. To apply the architecture it proposes two basic technology mappings [18] based on widely used standards in Industry 4.0 such as HTTP (RESTful) [21] and OPC-UA [22], which accepts MQTT [23] for PubSub. These three technologies are based on the TCP/IP protocol and belong to the so-called Industrial Internet of Things (IIoT) ecosystem. MQTT is the most popular PubSub protocol for the IoT and IIoT.

2.2. Digital Twin

As introduced earlier, a Digital Twin is a virtual reproduction of a system based on simulations, real-time and historical data that allows representing, understanding, and predicting scenarios of the past, present, and future, with verified and validated models, and synchronized at a specified frequency and fidelity with the system [9].

The two constituent parts of the Digital Twin are the Digital Model (DM) and the Digital Shadow (DS) [24]. On the one hand, the DM comprises the requirements, specifications, and theoretical models, both assets and asset simulators. On the other hand, the DS contains models based on data captured from the actual world, via observation or by automatic measurements using sensors

In addition to the DM and DS model, Stark [25] introduces the concept of the Digital Master (DM), which is defined as the set of digital models used in the Digital Twin. Hence, the formula DT = DM + DS expresses that Digital Twins are composed of models and data. Both DM and DS, and DT can be modeled as I4.0 Components [9]. According to Drath [26], Digital Twins can be defined based on I4.0 Components.

Moreover, the expression that a Digital Twin is composed of simulators and data is valid in most cases. DT synchronization can be achieved by the cyclical calibration and adjustment of the simulators (DM) from the captured data in the DS. Another aspect of synchronization relies on the execution speed of simulations. For complex models that simulations require a long time, model order reduction techniques are key for Digital Twins [27].

Industry 4.0 highly demands interdisciplinary approaches [28], and the Digital Twin is becoming a core element of model-based systems engineering (MBSE) [29]. Hence models must be conceptualized using formal languages involving several specialists or experts of diverse disciplines, and latter codified utilizing programming languages. According to Lindemann [30], the Digital Twin can encapsulate the interdisciplinary models of an asset, and it has been identified as a crucial artifact for implement Industry 4.0 [26].

The notion of Minimum Viable Digital Twin (MVDT) has been presented by Schalkwyk [31] initially in the Industrial Internet Consortium, and afterward in the Digital Twin Consortium. It is inspired by the principles of the Minimum Viable Product (MVP) proposed by Ries [32], based on Blank product development concepts [33]. According to Thomson, [34], the success of an MVP validates an idea, but its failure does not invalidate it. Schalkwyk [31] proposes the MVDT as a DT implemented at the conceptualization phase as a starting point for interdisciplinary approaches. DT models have the capability to encapsulate tacit and explicit knowledge [35], and both can be conceptually validated and transferred.

Madni [29] introduces the first level of a Digital Twin naming it pre-Digital Twin, in which the system may not yet exist, and there is no need for DS. It supports decision-making at the concept design and preliminary design.

According to Sargent [36], mutually accepted models can be achieved at the early stages with face-to-face or Conceptual Model validation using formal specification languages. The starting point can be building an MVDT, implementing simulators from scratch (pre-Digital Twin), or orchestrating and integrating existing simulators with Industry 4.0 low-code tools.

2.3. Low-Code Programming

The term low-code programming was coined in a 2014 Forrester Research report [37], and is regarded as a paradigm that leverages visual tools to accelerate software development by dramatically reducing the amount of hand-coding required [37]. The initial purpose was to run projects at "digital clock speed", a term introduced by Fine [38] that denotes a characteristic pace that organizations have. Nowadays, we have an additional purpose: breaking silos.

Existing low-code tools are both oriented to programmers and non-programmers, enabling the quick generation and delivery of applications with minimum effort by requiring the least possible effort to install and configure the development environments, as well as the time required for training and developing [39]. Furthermore, low-code programming enables people to get an effective digital cultural literacy, effectively democratizing software development [40].

Low-code usually relies on visual programming and, according to Mason [41], they can be divided into two broad categories: flow-based and block-based. On the one hand, flow-based are functional languages, and they are reasoned about as data flowing from one node to another. On the other hand, block-based are imperative languages, and they are reasoned as mutating state. Examples of flow-based low code environments include those based on BPMN (Business Process Model Notation) [42], such as those are in the BPMN Tool Matrix list [43]. In contrast, examples of block-based programming languages include Logo Blocks, Scratch or Blockly [44].

Block-based low-code imperative programming languages usually focus on programming rather than modeling.

They permit exploring, creating, and remixing interactive animations, supporting structured, modular, object-oriented, agent-oriented, and parallel programming paradigms. However, they do not directly support formal modeling elements for state machines, signaling, or discrete event simulations. Nevertheless, such limitation can be circumvented by the use of library extensions, which allow to add new blocks mapped to a formal specification language, such as SDL (Specification and Description Language).

3. Tools and Mapping to build SDL4Snap!

In this section we introduce the SDL modeling language and the Snap! programming language, and present how to map SDL on to Snap! to create the Snap!4SDL tool.

3.1. The SDL Modeling Language

A modeling language is any language that can be used to express information, knowledge, or systems by using a consistent set of rules. Modeling languages can be graphical or non-graphical and executable or non-executable.

SDL is a well-known standardized modeling language with an easy-to-understand graphical representation that is formally consistent (unambiguous), complete, and executable [35]. Given its characteristics, SDL is widely used in academia and in the industry to build models for simulation software, process control, real-time applications, and telecommunications systems. SDL development tools such as SDLPS [45] or PragmaDev Studio [46] are focused on manage complex SDL specifications.

SDL models can be converted to Discrete Event System Specification (DEVS) [47], which is considered a universal reference formal model for simulation engines [48]. In addition, DEVS models can be converted to SDL models [49] which, in turn, can be translated into state machine-based algorithms [50]. Therefore, DEVS models can be translated into a general-purpose programming language.

SDL models can also be formally verified and automatically converted to executable artifacts, such as simulators, which capture both explicit and tacit knowledge, so SDL is a way to represent tacit knowledge [35].

In addition, SDL can be used stand-alone or integrated with the Unified Modeling Language (UML) [51] as an UML Profile [52]. System, block, block class, process, and process class elements have been defined as UML stereotypes. In addition, an I4.0 Component can be defined as an UML component stereotype.

Last but not least, SDL can also be combined with 2D or 3D synchronized animations of the models, calling a user defined procedure call for this purpose in the right state and event of the SDL specification [35].

The SDL architecture and its building blocks are shown in Figure 3. An SDL system is composed of block agents or block processes. SDL is conceptually distributed, and agents can communicate with each other via messages (also called signals). A process is an agent and has a message queue that receives signals from other agents, can execute algorithms, and has the capability to send messages to other agents. The SDL engine is the system that executes the simulation, determining the order of the events to be treated by the processes, as depicted in Figure 4.

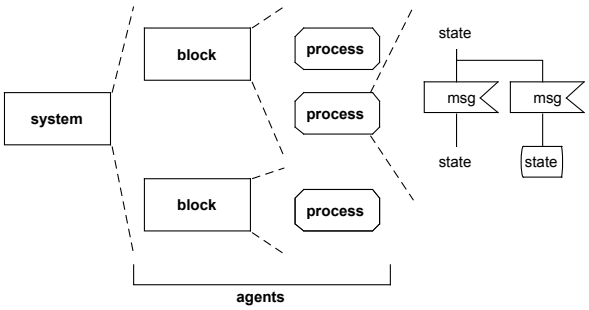


Figure 3. SDL architecture and building blocks

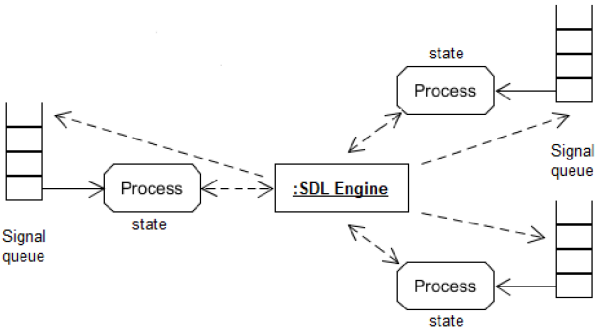


Figure 4. SDL signal queues

At the process level, SDL represents states and messages in a diagram that combines finite state machines, flowchart decision logic, and message sending and receiving specifications. This unified representation is general enough for modeling the integration of existing models with co-simulation [45], which consists of executing the subsystems simulations separately, with adequate coordination between them.

3.2. The Snap! Programming Language

Snap! is a graphical blocks-based programming language created by Jens Mönig and Brian Harvey in 2011. Before becoming a browser application (i.e., no software installation needed), the Snap! project's former name was BYOB (Build Your Own Blocks). Currently, the Snap! project is driven by the University of California Berkeley and the SAP software company. While inspired by Scratch, Snap! has the feature of user-defined JavaScript blocks, which permit extraordinary extension possibilities.

The main building block of Snap! is the block, which can be a command (i.e., procedure) or a reporter (i.e., function). Reporters are intended for synchronous calls, and commands support asynchronous calls by using callback blocks. Snap!

also allows to implement blocks directly in JavaScript, which empowers the development of library extensions.

A higher level fundamental building block in Snap*!* is the sprite, which is used to manage its looks and sounds. Sprites support object orientation, with properties, operations and parallel execution. In addition, sprites also support agent orientation with the capability of sending and receiving messages.

We adopt the Wooldridge weak notion of agent defined as a hardware/software system with the properties of autonomy, reactivity, proactivity, and social ability [53] to characterize both SDL and Snap*!* agent orientation. Sprites are agents that share a singleton object called stage, which defines a scope for a set of sprites. Other agents are the Snap*!* browser tabs that have their correspondent stages. One browser can run simultaneously many Snap*!* instances in different tabs, and one computer can run simultaneously many browser instances.

The "send" and "broadcast" native Snap*!* blocks can send messages to sprite, and other sprite blocks can be invoked remotely with the "tell" and "ask" blocks for commands and reporters, respectively. Sending and receiving messages from/to different Snap*!* instances can be achieved with extensions like MQTT4Snap*!*, interacting each other under a publish-subscribe architecture[1].

3.3. Mapping SDL on to Snap*!* to create the SDL4Snap*!* extension

In this section, we present a library extension to include SDL model-level abstraction support to Snap*!*, explaining how SDL converts to pseudocode and how the implemented SDL elements subset map to the Snap*!* blocks, which rely on the SDL Engine, also implemented in Snap*!*.

To translate SDL into pseudocode in SDL4Snap*!* we use the scheme suggested by Rockström and Saracco [54]. Figure 5 shows the translation of Algorithm 1 into Snap*!*. As it can be observed, the algorithm is based on two nested switch structures, where the external one branches by the state value and the branches of the internal one by the signal received. Unexpected signals are ignored, but SDL4Snap*!* could easily be modified to warn or halt if required.

To implement SDL in Snap*!*, we have adopted the logic of SDL processes message queues described by Belina [55]. That is, each agent (i.e., a sprite that corresponds to an SDL process) has two attributes: SDL signal input queue and SDL state, as shown in Figure 4. Sprites, as agents, can send and receive messages in both local or network modes. Extensions like MQTT4Snap*!* allow messages to go over a local network and also over the Internet, allowing sprites to send and receive messages from/to different browsers. Moreover, so approach allows emulating both synchronous and asynchronous calls in a client-server architecture, as depicted in Figure 1. Based on cooperating browsers, multiple distributed SDL blocks can constitute SDL systems, as illustrated in Figure 3.

1. MQTT4Snap*!* is an open-source Snap*!* MQTT extension library available in a public GitHub repository (https://github.com/pixavier/mqtt4snap.

Algorithm 1

```
switch (state)
  case StateA:
    switch (signal)
      case msg1:
          action1
      case msg2:
          action2
      default:
          Unexpected signal => halt, warn or ignore
    end switch
  case StateB:
    switch (signal)
      case msg1:
          action1
      default:
          Unexpected signal => halt, warn or ignore
    end switch
end switch
```

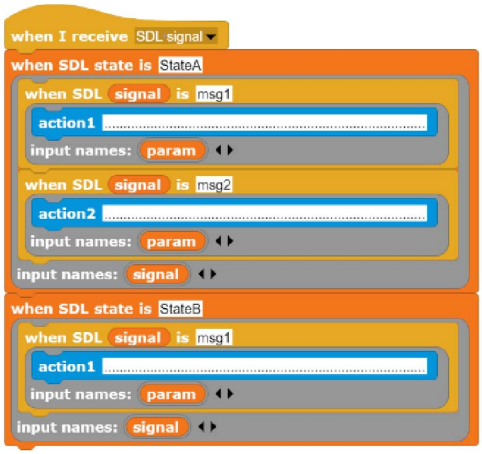


Figure 5. Algorithm 1 and its translation into Snap*!*.

SDL4Snap*!* considers the minimum set of SDL elements to implement a minimal consistent and complete SDL engine, as identified by Fonseca [56]: Start, State, Input, Create, Task, Output, Decision, Set State and Block.

The element-by-element SDL to Snap*!* mapping is explained textually below, but the Table 1 shows graphically the mapping at a glance.

The SDL "start" element maps onto the "when green flag clicked" and "SDL start" blocks. The last one sets up the process (agent), initializing its "SDL signal input queue" and "SDL state" attributes. The SDL "state" element maps onto an event Snap*!* block that fires when the SDL Signal event is received, combined with "when SDL state is ..." with a callback block. The SDL "input" element maps onto the "when SDL signal is ..." with a callback block. The SDL "create" element, restricted to processes, maps

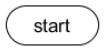
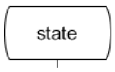
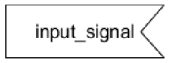
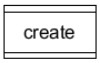
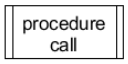
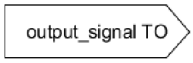
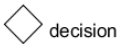
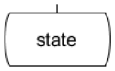
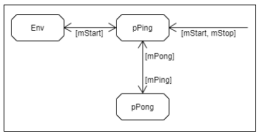
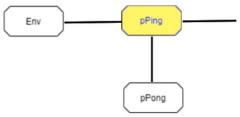
Element	SDL	Snap!
Start		
State		
Input		
Create		
Task		
Output		
Decision		
Set State		
Block		

Table 1. SDL ELEMENTS MAPPING ONTO SNAP! ELEMENTS.

onto the create Snap! block. The SDL task (procedure call) element has maps onto the standard Snap! block. The SDL "output" element maps onto the "SDL send local signal ..." or the "SDL send signal ..." block, and it supports parameter passing. This parameter can be a JSON (JavaScript Object Notation) string. The SDL decision element maps to the "if-then-else" standard block of Snap!. The SDL "set state" maps onto "SDL set state", and it is always present at the end of SDL threads.

The combination of SDL and Snap! is possible in both ways. On the one hand, we can build an SDL model and enrich it with user-defined procedures, and on the other hand, a Snap! project can use SDL to define the behavior of some specific elements, such as sprites.

The SDL elements shown in Figure 3 can also be mapped onto Snap!. The SDL processes map onto Snap! sprites, whereas the SDL blocks map onto the Snap! stages. Please notice that several stages can be simultaneously active in various browser tabs or independent browsers instances in the same or different computers. As mentioned earlier, signals can travel between stages using the MQTT4Snap! extension.

4. An SDL4Snap! implementation example: PingPong

In this section we present a minimal SDL4Snap! example, namely PingPong, and demonstrate how to convert it into an I4.0 Component.

4.1. PingPong SDL4Snap! implementation

PingPong is a basic multi-agent example that comes out of the box with SDL tools, and can be regarded as a "Hello World" program. The PingPong example implies two agents exchanging messages between them.

In our implementation using SDL4Snap!/[2], there are two agents (i.e., SDL processes), called pPing and pPong. A third auxiliary agent called Env represents the environment, and its only purpose is to trigger the initial signal to start the message exchange. The subsequent messages are sent alternatively from pPing to pPong and vice-versa.

The block diagram of the SDL model (top) and the SDL behavior of the pPing agent (bottom) are depicted in Figure 6. The process to convert the former to the latter can be regarded as an example[3] of the mapping rules shown in Table 1. As it can be observed, the agent pPing can be in two states: idle and running. The idle state is the default one when the agent starts, and in this state, the agent can receive the mStart signal. When the agent is in the "running" state, it can receive one of the following three signals: mPong, mStop, and tWait. Hence, the possible transitions are "idle" to "running" and "running" to "idle", as shown in the diagram. Finally, the user-defined procedure call for animation purposes is named "animTo", and it can use the graphical and multimedia Snap! potential.

4.2. PingPong as an Industry 4.0 Component

The PingPong example presented in the previous section can be easily encapsulated into an I4.0 Component to show how to formalize low-code components with RAMI 4.0 at the architectural level, using UML and SDL at the modeling level, and using Snap! at the implementation level.

The component specification is depicted in Figure 7. It includes the "digital bell" of the I4.0 Component, which only exposes added value digital services while hiding the internal component complexity. It consists of the PingPong facade class with its attributes and, PubTopic and SubTopic enumerations with PubSub defined operations by their correspondent topics. As shown, the system can do the following operations: start, stop and notify each time a process agent is animated. The notation shown in Figure 7 is a simplification of the UML lollipop representation of the graphically stereotyped I4.0 Component for a pure PubSub component, as depicted in Figure 8.

Using RAMI 4.0 terminology, we apply an MQTT technology mapping, implementing the specification shown in Figure 7. To allow the I4.0 Component to be accessible ubiquitously, we use the MQTT4Snap! extension presented earlier. In the Env agent, which is the initiator of the execution, the component subscribes to the "pingpong/start" and "pingpong/stop" topics. According to the SDL specification depicted in Figure 6, when a start message is received, a local SDL signal mStart is sent to pPing, and when a stop message is received, a local SDL signal mStop is sent to pPing. The resulting code is shown in Figure 9.

The third and last operation to implement is the publication of the animation of each process agent, informing the agent ID (pPing or pPong) and a timestamp. For that, we have to add the line shown in Figure 10 at the end of the "anim to" block. This "notify" operation can delegate the PingPong animation to a remote agent that subscribes to it.

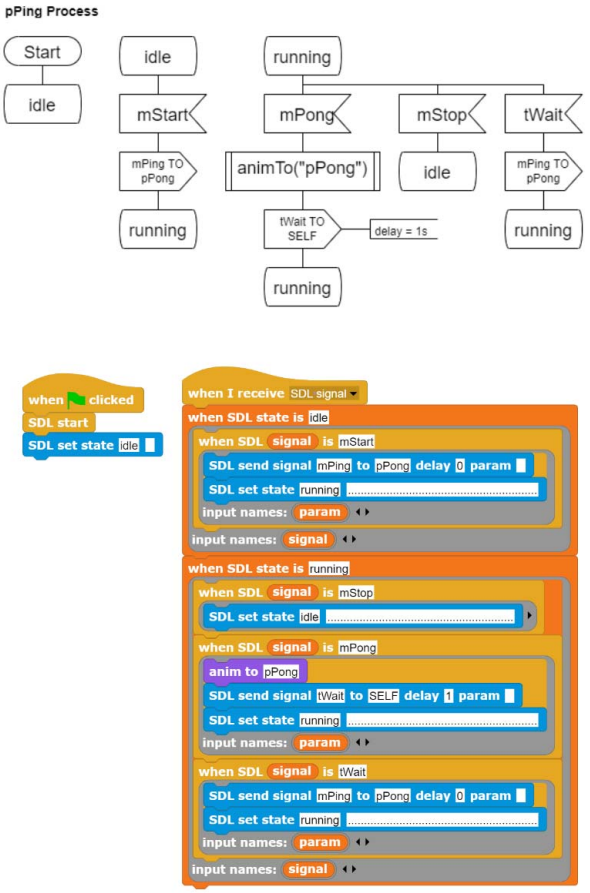


Figure 6. pPing agent modeled in SDL and mapped onto Snap!

A set of Snap! independent blocks shown in Figure 11 has been used to test the resulting PingPong I4.0 Component from another Internet browser instance. A single click on each can execute these Snap! blocks independently. After clicking on the connection block, we click on the start block, check that PingPong starts, click on the stop block, and check that PingPong stops. Finally, there is a subscription block to check that we receive the animation notifications.

Last but not least, it is important to mention that SDL4Snap! can also be used as a simplified facade of more complex DTs. It can interact with other low-code tools, such as Node-RED, enabling easy interaction with existing Industry 4.0 systems through their communication protocols. On the other hand, it can interact with instant messaging systems, such as Telegram, to explore new use cases such as integrating I4.0 Components with bots.

2. The PingPong example is available in the SDL4Snap! public GitHub repository (https://github.com/pixavier/sdl4snap)). It can be executed online, and all source code and technical details are also provided.

3. Please notice that the "SDL set state" rightmost dummy-parameter filled with dots is only added for formatting purposes.

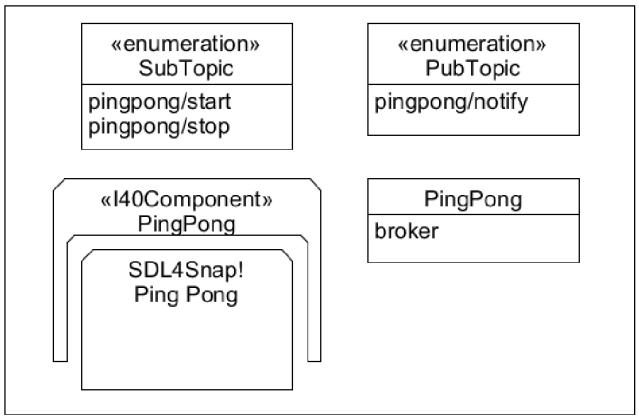


Figure 7. PingPong as UML I4.0 Component.

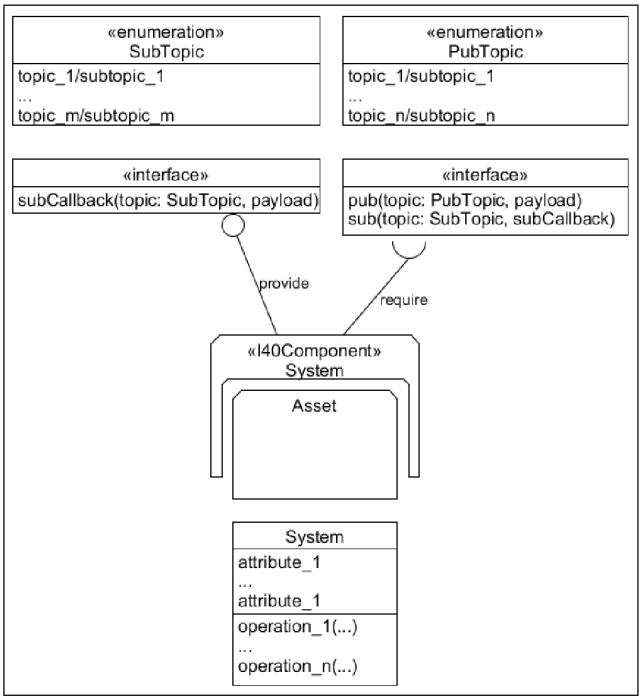


Figure 8. I4.0 Component as stereotyped UML Component

5. Preliminary evaluation of SDL4Snap*!*

To understand the potential of the SDL4Snap*!* tool, we have conducted a preliminary evaluation in the context of Industry 4.0 postgraduate programs at Universitat Politècnica de Catalunya (UPC) and Universitat Oberta de Catalunya (UOC). In these courses, students are introduced to the concept of Digital Twins and are required to put it into practice by studying how different self-contained examples of industrial process automation, either discrete or continuous, are implemented and testing how they operate. For instance, we provide Digital Twin models of water

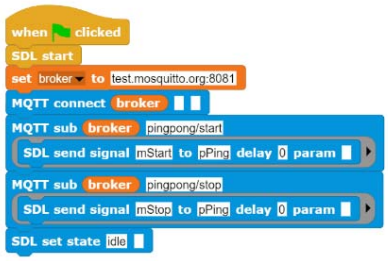


Figure 9. I4.0 Component Snap*!* implementation

Figure 10. I4.0 Component Snap*!* notifications publisher implementation

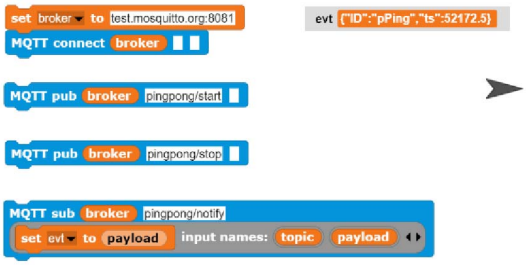


Figure 11. I4.0 Component Snap*!* testing blocks

treatment processes, bottling-packaging processes, and an elevator system.

Students of these programs have an engineering background, either from the IT or the OT worlds. It is an important factor to consider, as students with an OT background have had less exposure to programming. In contrast, students with an IT background have had less exposure to process automation. Thus, creating a Digital Twin represents a challenge to both, as their understanding of the industrial process automation or the tools to implement it are limited.

During the initial versions of the courses, the implementation of the Digital Twin models was based on textual programming languages, such as Python, JavaScript, or C/C++. While the provided examples were considered valuable to understand the concept and applicability of Digital Twins, we observed that a significant part of the students with an OT background found that the learning curve to understand the implementation was disproportionate. Moreover, such a steep learning curve caused frustration and, in some cases, limited the motivation to continue learning.

In contrast, after introducing SDL4Snap*!* in later versions of the same courses, we observed that these problems vanished thanks to the low entry barrier of block-based visual programming tools. Moreover, we also observed that the learning curve was also shortened in later activities, consisting of converting parts of these Digital Twin models to textual programming languages. We believe that this may

be related to the fact that students better comprehend the problems and the tools. It is also important to mention that such low-code tools did not seem to create an aversion to students with an IT background.

While we acknowledge that only qualitative data has been gathered so far, these early results show that following a low-code approach can increase student engagement in the activities, as well as their satisfaction with the learning outcomes, when compared to using non-low-code programming languages for similar activities. Hence, SDL4Snap*!* can be used as a resource to teach the modeling process of both continuous and discrete industrial systems and become a facilitator in the adoption of the Digital Twin within the industrial fabric.

6. Conclusions and Future Work

In this article, we have presented SDL4Snap*!*, an open-source and web-based tool that combines the SDL modeling language and the Snap*!* low-code block-based programming language. SDL4Snap*!* facilitates to design, build, test, and use of Digital Twin models that are compliant with the SDL specification. To evaluate the suitability of SDL4Snap*!*, we have conducted a preliminary qualitative evaluation in the context of Industry 4.0 postgraduate courses. The results show that SDL4Snap*!* has the potential to reduce the entry barrier to digital technologies, thus fostering their adoption in the context of Industry 4.0.

Taking into account these results, in the future, we plan to create an open repository of Digital Twin models compatible with SDL4Snap*!*, allowing students to use it for learning purposes. In addition, we also plan to extend the evaluation process to gather quantitative data that enables us to measure to which extent the introduction of SDL4Snap*!* improves student engagement and learning outcomes.

References

[1] H. Kagermann, W. Wahlster, and J. Helbig, "Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry," ACATECH – National Academy of Science and Engineering, Munchen, Final Report of the Industrie 4.0 Working Group, apr 2013.

[2] IEC, "Iec pas 63088:2017 smart manufacturing - reference architecture model industry 4.0," IEC (Standard, International Electrochemical Commission), Tech. Rep., 2017.

[3] Plattform-Industrie-4.0, "Exemplification of the industrie 4.0 application scenario value-based service following iira structure," Plattform Industrie 4.0, VDI and VDE, Tech. Rep., 2017.

[4] J. Sifakis, S. Bliudze, S. Furic, and A. Viel, "Rigorous design of cyber-physical systems: Linking physicality and computation," Software & Systems Modeling, 12 2017.

[5] E. Prieto-Araujo, P. Olivella-Rosell, M. Cheah-Mañe, R. Villafafila-Robles, and O. Gomis-Bellmunt, "Renewable energy emulation concepts for microgrids," Renewable and Sustainable Energy Reviews, vol. 50, pp. 325–345, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032115003718

[6] T. Ohnemus, "The digital twin – a critical enabler of industry 4.0," Zeitschrift für wirtschaftlichen Fabrikbetrieb, vol. 115, no. s1, pp. 23–25, 2020. [Online]. Available: https://doi.org/10.3139/104.112308

[7] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," Michael W. Grieves, LLC Whitepaper, 2014.

[8] M. Grieves and J. Vickers, Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. Cham: Springer International Publishing, 2017, pp. 85–113. [Online]. Available: https://doi.org/10.1007/978-3-319-38756-7_4

[9] P. Fonseca i Casas, J. Garcia i Subirana, V. García i Carrasco, and X. Pi i Palomés, "Sars-cov-2 spread forecast dynamic model validation through digital twin approach, catalonia case study," Mathematics, vol. 9, no. 14, 2021. [Online]. Available: https://www.mdpi.com/2227-7390/9/14/1660

[10] E. Commission, "Industry 4.0 for SMEs - smart manufacturing and logistics for SMEs in an x-to-order and mass customization environment," SME 4.0 - Horizon 2020 CORDIS Project- European Commission, Tech. Rep., 2019. [Online]. Available: https://cordis.europa.eu/project/id/734713/reporting/es

[11] V. de Leeuw, "Creating and deploying digital twins in the process industries," ARC Advisory Group, Tech. Rep., Apr. 2019. [Online]. Available: https://www.arcweb.com/blog/creating-deploying-digital-twins-process-industries

[12] S. Datta, "Emergence of digital twins - is this the march of reason?" Journal of Innovation Management, vol. 5, p. 14, 11 2017.

[13] J. Cabot, "Positioning of the low-code movement within the field of model-driven engineering," in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3417990.3420210

[14] R. Sanchis, O. Garcia-Perales, F. Fraile, and R. Poler, "Low-Code as Enabler of Digital Transformation in Manufacturing Industry," Applied Sciences, vol. 10, no. 1, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/1/12

[15] E. Modrow, Computer Science with Snap! emu-online Scheden, 2018. [Online]. Available: http://www.emu-online.de/ComputerScienceWithSnap.pdf

[16] VDI, VDE, and ZVEI, "Reference architecture model industrie 4.0 (rami4.0)," VDI/VDE Gessellschaft, Tech. Rep., Jul. 2015.

[17] P. I. 4.0, "Details of the administration shell. from idea to implementation," Plattform Industrie 4.0, Tech. Rep., 2018.

[18] VDI and VDE, "Industrie 4.0 service architecture basic concepts for interoperability," VDI/VDE Gessellschaft, Tech. Rep., Nov. 2016.

[19] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005.

[20] T. Erl, SOA Design Patterns. Prentice Hall PTR, 2009.

[21] H. Ed-douibi, J. L. C. Izquierdo, A. Gómez, M. Tisi, and J. Cabot, "Emf-rest: Generation of restful apis from models," in Proceedings of the 31st Annual ACM Symposium on Applied Computing, ser. SAC '16. New York, NY, USA: ACM, 2016, pp. 1446–1453.

[22] OPC-Foundation, "OPC Unified Architecture - Pioneer of the 4th industrial (r)evolution," OPC Foundation, Tech. Rep., 2014.

[23] ISO Central Secretary, "ISO/IEC 20922:2016 Information technology - Message Queuing Telemetry Transport (MQTT) v3.1.1," International Organization for Standardization, Geneva, CH, Standard ISO/IEC 20922:2016, 2016. [Online]. Available: https://www.iso.org/standard/69466.html

[24] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," IFAC-PapersOnLine, vol. 51, no. 11, pp. 1016 – 1022, 2018, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.

[25] R. Stark, S. Kind, and S. Neumeyer, "Innovations in digital modelling for next generation manufacturing system design," CIRP Annals, vol. 66, no. 1, pp. 169 – 172, 2017.

[26] R. Drath, "The digital twin - the evolution of a key concept of industry 4.0," Fraunhofer IOSB - [Industrial IoT - Digital Twin], pp. 8–9, 2018.

[27] D. Hartmann, M. Herz, and U. Wever, Model Order Reduction a Key Technology for Digital Twins. Cham: Springer International Publishing, 2018, pp. 167–179. [Online]. Available: https://doi.org/10.1007/978-3-319-75319-5_8

[28] VDI and ASME, "A discussion of qualifications and skills in the factory of the future: A german and american perspective," VDI/ASME, Tech. Rep., Apr. 2015.

[29] A. M. Madni, C. C. Madni, and S. D. Lucero, "Leveraging digital twin technology in model-based systems engineering," Systems, vol. 7, no. 1, 2019. [Online]. Available: https://www.mdpi.com/2079-8954/7/1/7

[30] Lindemann, Benjamin and Talkhestani, Behrang Ashtari and Jung, Tobias and Sahlab, Nada and Jazdi, Nasser and Schloegl, Wolfgang and Weyrich, Michael, "An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System," at - Automatisierungstechnik, vol. 67, no. 9, pp. 762 – 782, 2019. [Online]. Available: https://www.degruyter.com/view/journals/auto/67/9/article-p762.xml

[31] P. v. Schalkwyk, "The ultimate guide to digital twins," XMPro, Tech. Rep., 2019.

[32] E. Ries, The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown, 2011. [Online]. Available: https://books.google.es/books?id=tvfyz-4JILwC

[33] S. Blank, The Four Steps to the Epiphany: Successful Strategies for Products that Win. Lulu Enterprises Incorporated, 2003. [Online]. Available: https://books.google.es/books?id=oLL2pjn2RV0C

[34] T. Thompson, "Building a minimum viable product? you're probably doing it wrong," Harvard Business Review, 2013.

[35] P. Fonseca i Casas, X. Pi i Palomés, J. Casanovas, and J. Jové, Definition of Virtual Reality Simulation Models Using Specification and Description Language Diagrams. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–274.

[36] R. G. Sargent, "Verification and validation of simulation models," Journal of Simulation, vol. 7, no. 1, pp. 12–24, Feb 2013. [Online]. Available: https://doi.org/10.1057/jos.2012.20

[37] C. Richardson and J. R. Rymer, "New development platforms emerge forcustomer-facing applications," Forrester Research, Tech. Rep., Jun. 2014.

[38] C. H. Fine, "Industry clockspeed and competency chain design: an introductory essay," in Automation in Automotive Industries, A. Comacchio, G. Volpato, and A. Camuffo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 6–10.

[39] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," IFAC-PapersOnLine, vol. 52, no. 10, pp. 376–381, 2019, 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319309152

[40] R. D. Caballar, "Programming without code: The rise of no-code software development," Online IEEE Spectrum, Mar. 2020. [Online]. Available: https://spectrum.ieee.org/tech-talk/computing/software/programming-without-code-no-code-software-development

[41] D. Mason and K. Dave, "Block-based versus flow-based programming for naive programmers," in 2017 IEEE Blocks and Beyond Workshop (B B), 2017, pp. 25–28.

[42] OMG, Business Process Model and Notation (BPMN), Version 2.0, Object Management Group Std., Rev. 2.0, January 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[43] M. Hesse, "BPMN tool matrix," 2021, [Online; accessed 2021-07-19]. [Online]. Available: https://bpmnmatrix.github.io/

[44] M. Tempel, "Blocks programming," CSTA Voice, vol. 9, no. 1, Mar. 2013.

[45] P. Fonseca i Casas, "Enhancing sdlps with co-simulation," in Proceedings of the Winter Simulation Conference, ser. WSC '12. Winter Simulation Conference, 2012.

[46] PragmaDev, PragmaDev Studio Reference Manual. PragmaDev, 2018. [Online]. Available: https://www.pragmadev.com

[47] P. Fonseca i Casas, "Transforming SDL diagrams in a DEVS specification," in Proceedings MSO 2006, Sep. 2006.

[48] H. Vangheluwe, "DEVS as a Common Denominator for Multi-formalism Hybrid Systems Modelling, journal = IEEE International Symposium on Computer-Aided Control System Design," 08 2000.

[49] P. Fonseca i Casas, "Towards an automatic transformation from a DEVS to a SDL specification," in Proceedings of the 2009 Summer Computer Simulation Conference, ser. SCSC '09. Vista, CA: Society for Modeling & Simulation International, 2009, p. 348–355.

[50] D. Trossen, C. Cseh, and R. Kogan, Framework for Automatic SDL to C++ Translation. Boston, MA: Springer US, 1999, pp. 95–115. [Online]. Available: https://doi.org/10.1007/978-0-387-35578-8_6

[51] OMG, "Omg unified modeling language (omg uml). version 2.5.1," Object Management Group, Tech. Rep., 2017. [Online]. Available: http://www.omg.org/spec/UML/2.5.1

[52] ITU-T, "Z.109 : Specification and description language - unified modeling language profile for sdl-2010," ITU-T - TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, Tech. Rep., 2017.

[53] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," The Knowledge Engineering Review, vol. 10, no. 2, pp. 115–152, 1995.

[54] A. Rockström and R. Saracco, "Sdl - ccitt specification and description language," IEEE Transactions on Communications, vol. 30, pp. 1310–1318, 1982.

[55] F. Belina and D. Hogrefe, "The ccitt-specification and description language sdl," Computer Networks and ISDN Systems, vol. 16, no. 4, pp. 311 – 341, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0169755289900780

[56] P. Fonseca i Casas, "Using Specification and Description Language to define and implement discrete simulation models," in SummerSim '10 - 2010 Summer Simulation Multiconference, Ottawa, ON, Canada, July 11-14, 2010. Society for Computer Simulation International / ACM DL, 2010, pp. 419–426.