Minimum Viable Digital Twins with SDL and Snap*!*

Xavier Pi
Industry 4.0 Commission - Engineers of Catalonia,
Barcelona, Spain.
xpi@enginyers.net

Pau Fonseca
Universitat Politècnica de Catalunya - Barcelona Tech
(UPC), Barcelona, Spain.
pau@fib.upc.edu

ABSTRACT

SDL is a well known standardized graphical language, widely used for formal and non-ambiguous models for simulation software. It is easy to understand and it can be used by people coming from different disciplines. Snap*!* is an open-source graphical block-based programming language inspired by Scratch, that aims at students at any age and educational community to explore, create, and remix interactive animations. The MQTT4Snap*!* extension presented in this article lets Snap*!* implement distributed systems based on Publish-Subscribe (PubSub) architectures. In this paper we present a systematic and easy way to transform SDL models into Snap*!* programs. We also introduce its use for building Minimum Viable Digital Twins (MVDT), considering that the presented approach aims to shorten modeling and programming learning curves as much as possible. These MVDTs can be executed in both edge and cloud computing environments with zero license barriers, having the potential of integration with Internet of Things and Industry 4.0 solutions. Furthermore, an example of SDL to Snap*!* transformation is presented, which is available online for its detailed study.

1 INTRODUCTION

Taught postgraduate students of Industry 4.0 programs, come from the IT (Information Technologies) world, and the OT (Operation Technologies) world. They have an engineering background, and the majority part of those coming from OT considered disproportionate the effort in learning currently used programming languages for understanding and exercising Industry 4.0 concepts such as Distributed Architectures with the Internet of Things (IoT), I4.0 Components or Digital Twins.

Existing solutions include programming languages such as Python, JavaScript, C, or R, which can be combined with low-code programming languages like Node-RED. There are also modeling tools such as PragmaDev or SDLPS (based on the Specification and Description Language, SDL), InsightMaker (based on System Dynamics diagrams), engineering graphical modeling environments like MATLAB or Modelica, Business Process Modeling Language (BPMN) based tools or Petri Net oriented tools. All of them rely on formal methods that use diagram based languages, and they belong to the low-code programming languages family.

The use at the early stages in these postgraduate courses of traditional text-based programming languages, which have long learning curves, have been observed as a limiting learning factor because their learning process delays too much the hands-on activities on fundamental concepts of Industry 4.0. Moreover, modeling tools are not generally oriented to support co-simulation, so they cannot (or is difficult) interoperate between them. The following question arose: What are the programming and modeling languages with the shortest learning curves, suitable to exercise Industry 4.0 concepts such as Distributed Architectures with IoT, I4.0 Components, or Digital Twins?

Block (graphical) based languages developed at MIT MediaLab are the result of the research of finding the minimum viable programming languages, suitable to be used by children at an early age. So we considered them as the programming languages with the shortest learning curve. To implement I4.0 Components, IoT and Digital Twins Distributed Systems Architectures such as Client-server or Publish-subscribe (PubSub) must be supported. We propose to use low-code tools at the early stages of postgraduate courses of Industry 4.0, and carry out hands-on activities for all the involved concepts, achieving enough interoperability capabilities for implementing Minimum Viable Digital Twins.

The low-code block-oriented programming language Snap*!*, born as a conceptual fork of Scratch, was chosen because it includes a complete set of blocks and libraries. It is also extensible by using custom JavaScript blocks, which allow adding the support of the Message Queuing Telemetry Transport (MQTT) protocol, a widely used standard for IoT, that lets sending and receiving messages across the network (local or Internet), and enable the support of PubSub architectures. Our approach is to implement a library to map SDL models on to Snap*!*. Three contributions are presented. The first is the design and implementation of MQTT4Snap*!*, that enables Snap*!* to support distributed asynchronous messages. The second, based on the previous one, is the design and implementation of the SDL4Snap*!*, that lets direct mapping on to SDL models to Snap*!*. The third is the validation of the use of just low-code programming to hands-on fundamental concepts of Industry 4.0.

This paper is organized into five sections. Section 1 presents an introduction to the Industry 4.0 concepts of Distributed Architectures with IoT, I4.0 Components, and the concept of Digital Twin. Section 2 presents the SDL modeling language. Section 3 presents the Snap*!* programming language. Section 4 presents the mapping of SDL on to Snap*!* with SDL4Snap*!* (which is based on MQTT4Snap*!*). Finally, section 5 presents the concluding remarks.

2 INDUSTRY 4.0 CONCEPTS

The 4th Industrial Revolution, also known as Industry 4.0, is characterized by the Digital Transformation of the industrial fabric [1]. It is based on the marriage of the physical and digital worlds, which gives rise to the emergence of Cyber-physical Systems (CPS) [2]. The convergence of the worlds of IT and OT industrial phenomenon, known as "IT/OT Convergence", was identified by Rockwell Automation [3] in 2007, and it consists of both technological and cultural convergence. De Leeuw [4] has identified the existence of a minimum threshold of digital maturity in resources, systems, organization, and culture, required for a successful Digital Transformation process. We consider that the notions of Distributed Architectures with IoT, I4.0 Components, and the Digital Twin are

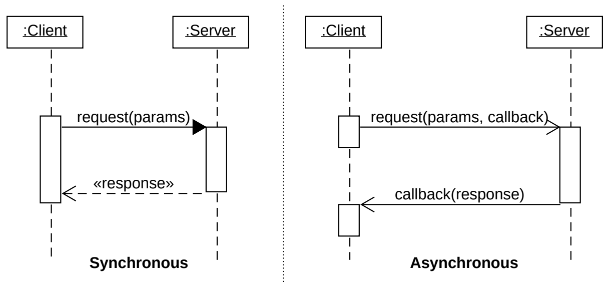


Figure 1: Client-server architectures

part of the minimum digital culture of the Industry 4.0 engineering professionals.

2.1 Distributed Architectures with IoT

The term Internet of Things emerged at the end of the nineties, and it can be defined as a global system of interconnected assets provided with a unique ID, with the capability of transfer data over a network. Assets can be modeled as intelligent agents, that are computational entities with the capabilities of autonomy, reactivity, pro-activity, and associativity.

In the IoT context, Client-server and Publish-subscribe (also known as PubSub) are the two main Distributed Systems architectures for the implementation of an agent-based Industry 4.0.

Client-server architectures are based on the request-respond mechanism, which has two variants, synchronous and asynchronous, as shown in Fig 1. The requester is a client role agent that makes a request directly to the responder. Whereas the responder is a server role agent that sends a response. An asynchronous request-response variant mechanism can be converted to synchronous using a semaphore (in this case, a software variable that changes when the response is received and let the requester program flow continue its execution)

PubSub architectures are not limited to one to one agents, and they are based on the broker mechanism, that can involve many agents. The publisher role agent sends a message to publish to a broker via a specific channel (topic) and all the subscriber role agents subscribed to this topic, receive the messages at the same time, as shown in Fig. 2. The broker is a common contact point in the network for publishers and subscribers. PubSub architectures have an asynchronous nature, but it is possible to implement an equivalent synchronous request-response behaviour if there is only one subscriber for the request and the publisher is also the only one subscriber of the response.

HTTP is the most popular protocol used for client-server solutions in an IoT context. It is used by Internet browsers and it was formally initially defined by the RFC 1945 IETF standard [5]. Currently, The most used variant of HTTP services is the so-called RESTful web services (Representational State Transfer), which are based on the requests that are independent of the previous ones.

MQTT is the most popular IoT PubSub oriented protocol, standardized as ISO/IEC 20922 [6]. The initial version was developed by

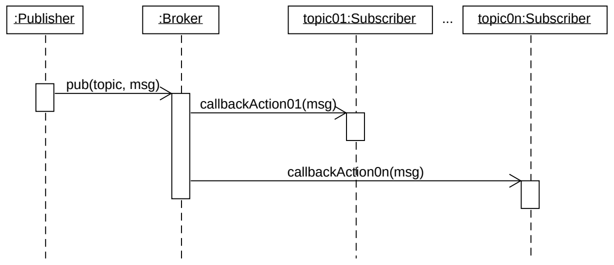


Figure 2: PubSub architectures

IBM in 1999, and in 2013 was submitted as an OASIS open standard. It is supported by many devices, industrial controllers, and Internet browsers.

OPC-UA [7] is another emergent protocol that is mainly oriented to industrial applications, and its last specification (IEC 62541) supports client-server and PubSub. It is technically easy to build gateways between these protocols using low-code tools like Node-RED.

2.2 I4.0 Components

The digitalization process of industrial physical assets is defined by the Reference Architecture Model Industry 4.0 (RAMI 4.0) [8], standardized as IEC/PAS 63088 [9]. RAMI 4.0 defines the notion of Administration Shell, which relays on the metaphor that every artifact (real or virtual) can be covered with a "digital bell", as shown in Fig 3, giving rise to the I4.0 Component. Therefore, it can be seen externally as a digital entity, that exposes an Application Programming Interface (API). From the moment of an asset is a full I4.0 Component, it can be remotely monitored and fully controlled across the network

An I4.0 Component has two main elements: The Component Manager and the Manifest [9].

The Component Manager consists of the specification of hardware and software to support computation capabilities (Turing Machine) and telecommunications. The standard specifies that I4.0 Components must support Service Oriented Architecture (SOA), providing an Application Program Interface (API) [9], which are

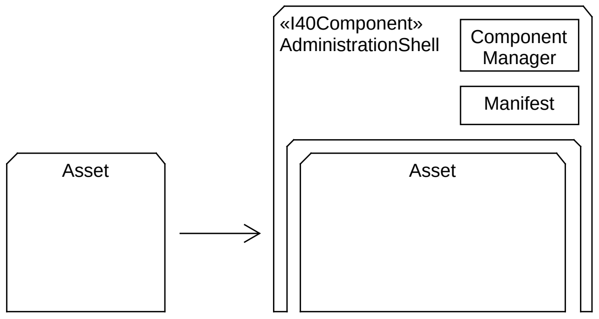


Figure 3: RAMI I4.0 Component

digital services that can be requested and delivered on a digital network. According to Herrera [10], multi-agent systems can be integrated into SOA based systems, hence they are supported by RAMI.

The Manifest consist of a set of structured data or pointers to the information that describes the asset and its context. This information can range from classical technical documentation to a set of simulation models. [9].

RAMI specifies that the I4.0 Components must have the characteristics of nestability and encapsulability. The first one means that every I4.0 Component can consist of other I4.0 Components which can be logically nested, including temporarily if needed. The second one means that each I4.0 Component should be able to establish all necessary connections within an I4.0 System, and they must be able to retain its core functionality even if the external network is disrupted [9].

RAMI is technologically agnostic, but it defines the notion of Technology Mapping to decide which technology must be used to implement an architectonic solution. OPC-UA, MQTT, and HTTP-RESTful are the default technology mappings options for I4.0 Components development [11].

Under RAMI 4.0, the CPSs are defined as networked compositions of I4.0 Components.

2.3 Minimum Viable Digital Twins

The concept of Digital Twin (DT) was introduced in 2002 by Michael Grieves [12], who stated that "the DT it is based on the idea that a digital informational construct about a physical system could be created as an entity on its own" [13]. Stark [14], from the Fraunhofer Institut, states that DT is based on two pillars, the Digital Master (DM) and the Digital Shadow (DS). On one hand, the DM is based on the specification and theoretical models, and it can be used to create physical assets and asset simulators. On the other hand, the DS is based on data collected from the real world, which can become a high volume of data (Big Data).

We have adopted the model proposed by Soler et al. [15], which is shown in Fig. 4 diagram, that represents a physical asset and its Digital Twin instance, composed by the Digital Shadow, simulators, and their traces.

The Digital Master is the origin of all elements, and it contains all the digital artifacts (documentation, models, algorithms, and data) that enable the creation of the physical asset and simulators models of several aspects of the asset. Physical assets and simulators are instances of the Digital Master. We also name physical assets instantiations as materializations. In Industry 4.0, production can be defined as multiple materializations of Digital Masters.

The Digital Shadow is the result of running the physical asset, that can be collected by sensors, SCADA systems or IoT, and also can be obtained by observations or expert's suggestions. On the other hand, traces are the result of running simulators. To build a DT, a common starting point is the Digital Master, usually coming from Engineering Departments, plus the collected data, or Digital Shadow. Hence the formula:

DT = DM + DS.

As time goes by, digital shadows and simulator's digital traces can be compared in order to calibrate or reprogram the simulators. If simulators must be always updated, the comparison and simulator's upgrade cycle must be executed continually.

The 4th Industrial Revolution is a Digital Revolution, all the elements of the DT are 100% digital, and on the other hand, physical assets are increasingly containing more digital elements.

The Digital Twin is a key concept of Industry 4.0 [16], and it is becoming a core element of model-based systems engineering (MBSE) [17]. Industry 4.0 highly demands interdisciplinary approaches [18] [19]. Hence models must be conceptualized using a formal language to involve several specialists or experts of diverse disciplines, and latter codified using a programming language. According to Lindemann [20], the Digital Twin can encapsulate the interdisciplinary models of an asset, and it has been identified as a crucial artifact for implement the Industry 4.0 [16]. Grieves [21] maintains that "Industry 4.0 is only possible with the Digital Twin".

The notion of Minimum Viable Digital Twin (MVDT) has been presented by Schalkwyk [22] initially in the Industrial Internet Consortium, and after in the Digital Twin Consortium. It is inspired by the principles of the Minimum Viable Product (MVP) proposed by Ries [23], based on Blank product development concepts [24]. According to Thomson [25], the success of an MVP validates an idea, but its failure does not invalidate it. Schalkwyk [22] proposes the MVDT as a DT implemented at the conceptualization phase.

In the Industry 4.0 postgraduate courses context, the MVDT must include a minimum simulator that produces traces comparable to a minimum given Digital Shadow and must be accepted by the involved stakeholders as the minimum mutually accepted model. This is the basis of our interdisciplinary approach.

According to Sargent [26], mutually accepted models can be achieved at the early stages with Conceptual Model validation using formal tools based on DEVS or SDL. The proposed way is to start building an MVDT, implementing the simulator with a suitable Industry 4.0 low-code tool, and compare simulation traces with an expected digital shadow that can come from observations or experts suggestions. MVDT is an executable conceptual model acting as software in the loop. DT Simulation models have the capability to encapsulate tacit and explicit knowledge [27], and both can be conceptually validated and transferred.

Our postgraduate courses experience related to hands-on activities with digital twins are based on several inputs. On one hand, we have digital shadows that come from IoT and data-analytics exercises. On the other hand, we must model the assets and build their simulators in order to complete the digital twin and experiment with it. The challenge is to joint all the minimum needed pieces as I4.0 Components as shown in Fig. 4, with the minimum effort in software coding.

We have chosen SDL as the formal modeling specification language to implement Minimum Viable Digital Twins, and Snap*!* as the programming language because both have the shortest learning curves.

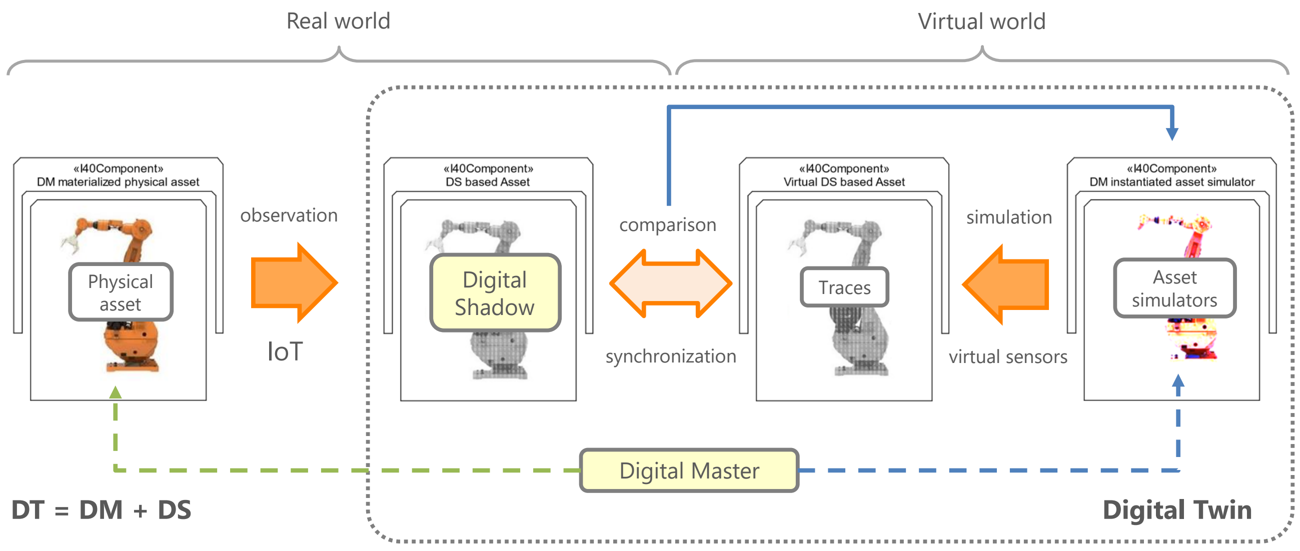


Figure 4: The concept of Digital Twin

3 THE SDL MODELING LANGUAGE

SDL is a system modeling language that has an easy to understand graphical representation, that is formally consistent (unambiguous) and complete, because their models can be converted to Discrete Event System Specification (DEVS) [28]. DEVS is considered the universal reference formal model for simulation engines [29]. DEVS models can be converted to SDL models [30] [31]. SDL models can be translated into a state machine-based algorithms [32]. Therefore they can be translated into a general-purpose programming language. There are several tools to run directly SDL graphic models, such as SDLPS or PragmaDev Studio, mainly oriented to professional users. In this article, we present the open-source SDL4Snap*!* [33] extension, that can be used by everybody, just with a standard Internet browser (no installations), at school, at home, at companies or the academia, without losing its formal character.

SDL is a well known standardized graphical language, widely used in the industry and in academia to build models for simulation software, process control, and real-time applications and telecommunications systems [34].

SDL is formally consistent (non-ambiguous) and complete. Hence SDL models can be formally verified and automatically converted to executable artifacts. SDL can be used stand-alone or integrated with the Unified Modeling Language (UML). System, block, block class, process, and process class elements have been defined as UML stereotypes. I4.0 Components can be defined as a UML component stereotype as well. SDL involves the notion of asynchronous message passing, finite state machines, and algorithms, and it can be combined with Publish-subscribe architectures to create co-simulation systems [35]. The SDL architecture and its building blocks are shown in Fig. 5. 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.

At the Process level, states and messages are intuitively represented in a diagram that combines finite state machines, flowchart decision models, and the message sending and receiving logic.

The coupling of heterogeneous simulation models has been identified by Schuh [36] as a crucial trend for the near future by the emergence of Digital Twins [37]. Simulations of systems composed of subsystems, designed and simulated by different teams require the coupling of several models.

The coupling of different models can be achieved by the use of co-simulations. They consist of executing the subsystems simulations separately, with adequate coordination between them. According to Sifakis [2], co-simulations explicitly discretizes the coupling signals, and it aims at decoupling the continuous dynamics of the subsystems wherever possible. The conversion from continuous to

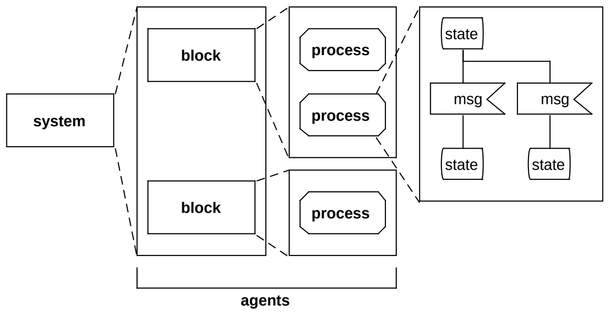


Figure 5: SDL architecture and building blocks

discrete can entail a loss of numerical stability resulting from the discretization. This effect can be generally addressed by choosing correct (usually small enough) values of delta time steps.

One of the hand-on activities proposed in Industry 4.0 postgraduate courses is to divide systems into subsystems, assign each subsystem to a team, model each subsystem independently building, and running the respective simulation models. And finally, couple the different models created by the teams using co-simulation, to get an entire system big simulation model.

4 THE SNAP*!* PROGRAMMING LANGUAGE

Cultural Literacy is the ability to understand and participate fluently in a given culture [38]. A Digital Cultural Literacy for everybody is an objective of the European Commission [39], and according to Resnick et al. [40], some type of programming is needed to achieve Digital Cultural Literacy. Low-code (or non-code) programming enables people to get an effective Digital Cultural Literacy, and it has been identified as an emergent trend [41] [42]. According to Gartner Inc [43], by 2024, low-code application development will be responsible for more than 65% of application development activity. Gardner's Magic Quadrant 2019 Report for Low-code Application Platforms (LCAP), points out that there are more than 15 companies offering solutions. The majority part is oriented to IT solutions (Microsoft, Betty Blocks, Mendix, among others) instead of OT solutions (Tulip). Graphical blocks-based languages are an example of Low-code programming languages. Examples of them are Scratch (https://scratch.mit.edu) and Snap (https://snap.berkeley.edu) on the cloud, and Snap4Arduino (http://snap4arduino.rocks) on the edge. Both Scratch and Snap*!* are open-source projects conceived to be a programming language "for everybody" [40]. The first one is a development of the MIT Media Lab available since 2007, and the second one is a Berkeley University and SAP company development, which is publicly available since 2011.

The Snap*!* Programming Language was born as a conceptual fork of the Scratch project, so they are very similar. It is also an open-source project, driven by Berkeley University and the SAP software company. Members of the academia, like Eckard Modrow [44], use Snap*!* in Computer Science courses, because it covers many aspects of the computer science such as structured, modular, object-oriented, agent-based, parallel or functional programming, among other characteristics. Moreover, it encourages to explore, create, and remix interactive animations.

Snap*!* is a graphical blocks-based programming language created by Jens Mönig and Brian Harvey in 2011. While inspired by Scratch, it has many advanced features. It does not require installation because the editor and programs created in it, run in the browser.

Snap*!* supports user-defined blocks, that can be implemented reusing existing blocks combined with blocks written in JavaScript (EcmaScript). Using this feature, the MQTT4Snap*!* [45] extension has been developed, providing the capability of distributed software over a Publish-subscribe.

The main building block in Snap*!* is the Sprite, that is prepared to manage their looks and sounds. Sprites support object orientation, with properties and operations, but they also support agent-orientation with the capability of sending and receiving messages.

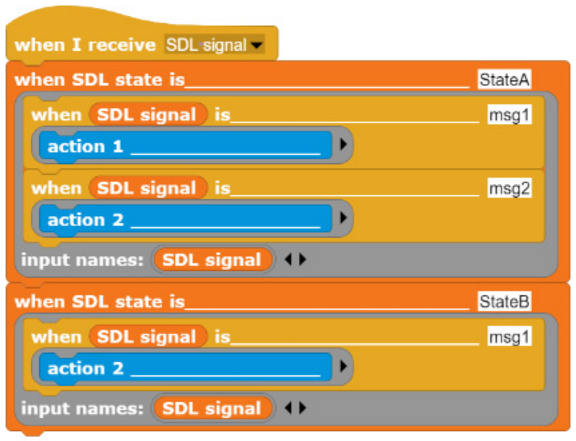


Figure 6: Algorithm 1 mapping onto Snap*!*

Sprites share a singleton object called Stage, that defines a scope for a set of sprites. Each Snap*!* browser instance has its stage and can correspond to the notion of SDL block

5 MAPPING SDL ON TO SNAP*!*

In this section we expose the subset of implemented SDL, the method of converting SDL to pseudocode, and the adoption of MQTT for managing distributed messaging.

SDL can be translated automatically to pseudocode, as suggested by Trossen [32] and Gaudin [46]. This scheme has been applied in SDL4Snap*!*. Fig. 6 depicts the translation of Algorithm 1.

The algorithm is based on two nested switch structures, where the external one branches by the state value and the internal ones branches by the signal received.

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
```

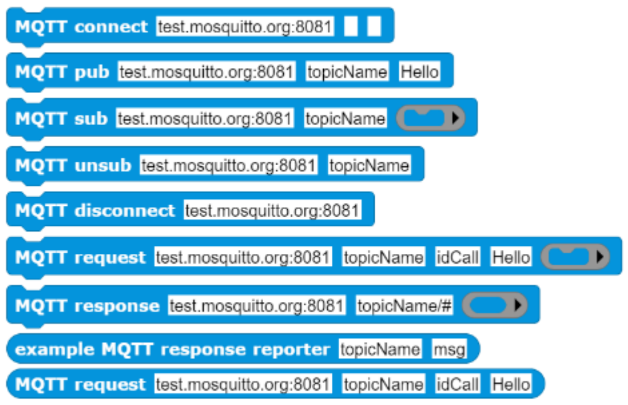

MQTT connect test.mosquitto.org:8081
MQTT pub test.mosquitto.org:8081 topicName Hello
MQTT sub test.mosquitto.org:8081 topicName
MQTT unsub test.mosquitto.org:8081 topicName
MQTT disconnect test.mosquitto.org:8081
MQTT request test.mosquitto.org:8081 topicName idCall Hello
MQTT response test.mosquitto.org:8081 topicName/#
example MQTT response reporter topicName msg
MQTT request test.mosquitto.org:8081 topicName idCall Hello

Figure 7: MQTT4Snap*!* library blocks

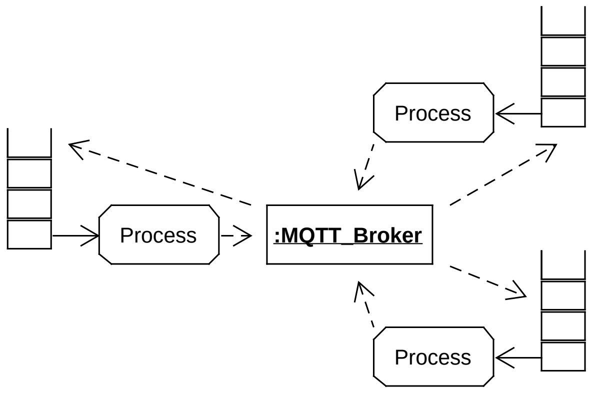


Figure 8: SDL queues on PubSub based architecture

5.1 MQTT4Snap*!*

The MQTT4Snap*!* is a Snap*!* software library that enables to use the MQTT protocol in the Snap*!* low-code programming language.

The supported primitives are:

- connect
- publish (pub)
- subscribe (sub)
- unsubscribe (unsub)
- disconnect
- request (synchronous)
- request (asynchronous)
- response

The library consists of nine blocks, shown in Fig, 7.

MQTT4Snap*!* is an open-source Snap*!* MQTT extension library that is available in a public GitHub repository [45].

5.2 SDL4Snap*!*

The logic of SDL processes message queues described by Belina [47] has been adopted, and it has been implemented over a PubSub architecture. Each agent, ie. sprite, (that corresponds to an SDL process) subscribes itself to its own topic, according to Fig. 8.

Sprites (agents) can send and receive messages in local mode (without network), or in network mode from/to localhost, over a local network and also over the Internet.

The technology chosen has been MQTT, and this feature relies upon MQTT4Snap*!*, the library presented previously.

This library lets sprites send and receive messages regardless of the browser in which the Snap*!* instance is running. Based on cooperating browsers, multiple distributed SDL blocks can constitute SDL systems, as depicted in Fig 5.

The minimum SDL elements to implement a consistent and complete SDL engine were identified by Fonseca [48], and they are the list of SDL mapped elements origin. The mapping between them and their correspondent Snap*!* blocks, shown in Table 1.

The SDL start element establishes the agent subscription to its corresponding topic, named by the agent ID.

The SDL state element has been mapped 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 has been mapped onto the "when SDL signal is", with a callback block.

The SDL create element has been mapped onto the create Snap*!* block.

The SDL task (procedure call) element has been mapped onto the standard Snap*!* block.

The SDL output element has been mapped onto the "SDL send local signal ..." or the "SDL send signal ..." block.

The SDL decision element has been mapped to the "if-then-else" standard block of Snap*!*. Other branching blocks provided by Snap*!* can be used.

The SDL processes are mapped onto Snap*!* sprites and the SDL blocks can be mapped onto the Snap*!* stage, but a single stage can include more than one block if we define a block as an arbitrary set of interconnected processes. In Table 1 there is the representation of the main block of the Ping-Pong example, on which the next section is based on.

5.3 A Mapping Sample

The Ping-Pong well-known example is considered as the "Hello World" of a multi-agent system. It consists of two agents, in our case modeled as two SDL processes, called pPing and pPong. There is also a third auxiliary agent called Env, that represents the environment, and its only purpose is to trigger the initial signal. The subsequent messages are sent alternatively from pPing to pPong and the reverse.

The complete example is fully implemented and available in the public GitHub repository of SDL4Snap*!* [33]. It can also be executed online just with one click, and all source code and technical details can be consulted on the fly.

The block diagram and the SDL behavior of the pPing agent are depicted in Fig 9. 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

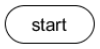
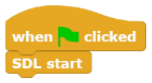
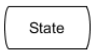
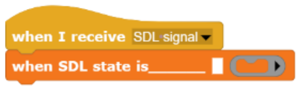
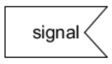
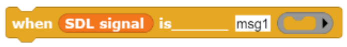
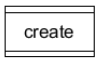
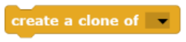
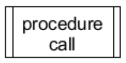
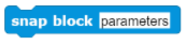
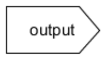
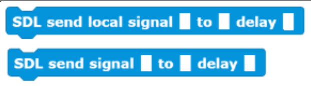
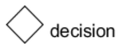
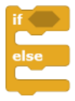
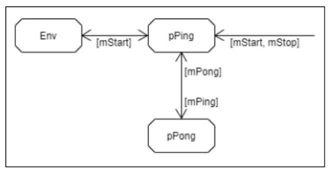
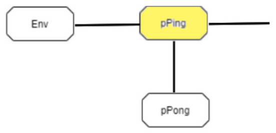
Element	SDL	Snap*!*
Start		
State		
Input		
Create		
Task		
Output		
Decision		
Block		

Table 1: SDL elements mapping onto Snap*!* elements

three signals: mPong, mStop, and tWait. The possible transitions shown in the diagram are idle to running, running to idle.

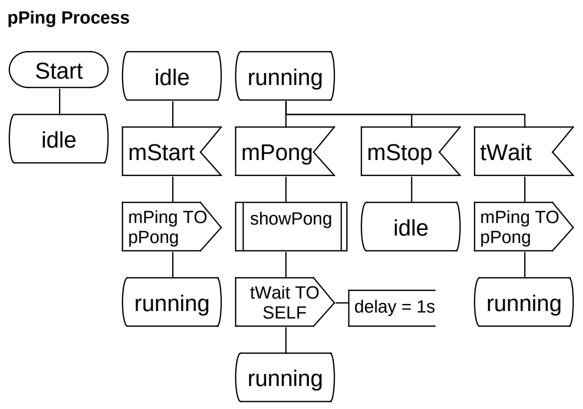
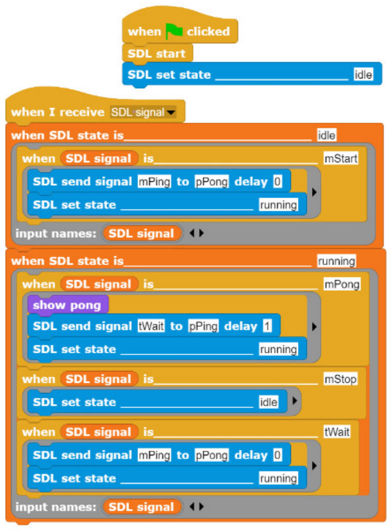


Figure 9: pPing agent in SDL



```
when [flag] clicked
SDL start
SDL set state idle

when I receive SDL signal
when SDL state is idle
  when SDL signal is mStart
    SDL send signal mPing to pPong delay 0
    SDL set state running
  input names: SDL signal
when SDL state is running
  when SDL signal is mPong
    show pong
    SDL send signal tWait to pPing delay 1
    SDL set state running
  when SDL signal is mStop
    SDL set state idle
  when SDL signal is tWait
    SDL send signal mPing to pPong delay 0
    SDL set state running
  input names: SDL signal
```

Figure 10: pPing agent in Snap*!*

The SDL model shown in Fig. 9 translation is depicted in Fig. 10. It is an example of mapping rules shown in Table 1 application.

6 CONCLUDING REMARKS

The combination of SDL, Snap*!*, and MQTT offers the possibility to implement a powerful low-code tool to build Minimum Viable Digital Twins. Both SDL and Snap*!* are the languages with the shortest learning curve for modeling and programming suitable for our purposes.

We have presented a ready to use an online open-source tool named SDL4Snap*!*, that enables us to build MVDTs. It uses the low-code Snap programming language, based on the SDL modeling language and a PubSub architecture, relying on the MQTT Internet of Things (IoT) messaging protocol. It can be used at any point in the asset life-cycle, even from the conceptualization phase, allowing non-specialists to participate in the modeling process, fostering interdisciplinary approaches.

The combination of SDL with Snap*!* has been used and evaluated in Industry 4.0 postgraduate courses at Universitat Politècnica de Catalunya (UPC) and Universitat Oberta de Catalunya (UOC). We have observed a qualitative improvement in the results of the Minimum Viable Digital Twins developed by the students in terms of agility and student satisfaction. Both continuous and discrete industrial systems have been modeled, such as water treatment processes or bottling and packaging processes.

We also have observed that in later activities consisting of converting part of these developments to other programming languages such as Python or JavaScript, the learning curve related to those languages has also shortened, because of the better comprehension of the problems, including software concepts.

As future work, the creation of an online executable examples repository is planned, with the potential to be linked to Industry 4.0 diagnostic tools. The application of these techniques in Business Schools to build interdisciplinary workshops for managers and engineers is part of future work.

The strong connection between the Snap*!* programming language to the primary and secondary school education, and also to the Maker community, is an opportunity to foster engineering studies vocations among the young people. It also opens the possibility to investigate the potential advantages of incorporating modeling skills (SDL mindset) at the early ages in the construction of a Digital Cultural Literacy in society.

REFERENCES

[1] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig. 2013. Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry. Final Report of the Industrie 4.0 Working Group. ACATECH – National Academy of Science and Engineering, Munchen.

[2] Joseph Sifakis, Simon Bliudze, Sebastien Furic, and Antoine Viel. 2017. Rigorous design of cyber-physical systems: linking physicality and computation. *Software & Systems Modeling*, (December 2017). DOI: 10.1007/s10270-017-0642-5.

[3] Rockwell-Automation. 2007. Come Together: IT-Controls Engineering Convergence Furthers Manufacturers. Technical report. Rockwell Automation.

[4] Valentijn de Leeuw. 2019. Creating and Deploying Digital Twins in the Process Industries. Technical report. ARC Advisory Group, (April 2019). https://www.arcweb.com/blog/creating-deploying-digital-twins-process-industries.

[5] Tim Berners Lee. 1996. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945. RFC Editor, 1–60. https://tools.ietf.org/html/rfc1945.

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

[7] OPC Foundation. 2014. OPC Unified Architecture - Pioneer of the 4th industrial (r)evolution. Technical report. OPC Foundation.

[8] VDI, VDE, and ZVEI. 2015. Reference Architecture Model Industrie 4.0 (RAMI4.0). Technical report. VDI/VDE Gessellschaft, (July 2015).

[9] IEC. 2017. IEC PAS 63088:2017 Smart Manufacturing - Reference Architecture Model Industry 4.0. Technical report. IEC (Standard, International Electrochemical Commission).

[10] V. V. Herrera, A. Bepperling, A. Lobov, H. Smit, A. W. Colombo, and J. L. M. Lastra. 2008. Integration of multi-agent systems and service-oriented architecture for industrial automation. In *2008 6th IEEE International Conference on Industrial Informatics*, 768–773. DOI: 10.1109/INDIN.2008.4618205.

[11] VDI and VDE. 2016. Industrie 4.0 Service Architecture Basic concepts for interoperability. Technical report. VDI/VDE Gessellschaft, (November 2016).

[12] Michael Grieves. 2014. Digital twin: manufacturing excellence through virtual factory replication. *Michael W. Grieves, LLC Whitepaper*. Retrieved 12/30/2017 from.

[13] Michael Grieves. 2016. Digital twin: mitigating unpredictable, undesirable emergent behavior in complex systems (excerpt). *Origins of the Digital Twin Concept (Working Paper)*, (August 2016).

[14] Rainer Stark, Simon Kind, and Sebastian Neumeyer. 2017. Innovations in digital modelling for next generation manufacturing system design. *CIRP Annals*, 66, 1, 169 –172. ISSN: 0007-8506. DOI: 10.1016/j.cirp.2017.04.045.

[15] Carles Soler and Xavier Pi. 2020. El concepte del "bessó digital" o Digital Twin. Working paper. Industry 4.0 Commission, Engineering Associations of Catalonia, Diagnostic 4.0 Workgroup (www.diagnostic40.com). https://diagnostic40.files.wordpress.com/2020/07/modelbessodigital_comissioindustria40-1.pdf.

[16] Rainer Drath. 2018. The digital twin - the evolution of a key concept of industry 4.0. *Fraunhofer IOSB - [Industrial IoT - Digital Twin]*, 8–9. ISSN: 1616-8240.

[17] Azad M. Madni, Carla C. Madni, and Scott D. Lucero. 2019. Leveraging digital twin technology in model-based systems engineering. *Systems*, 7, 1. ISSN: 2079-8954. DOI: 10.3390/systems7010007. https://www.mdpi.com/2079-8954/7/1/7.

[18] VDI and ASME. 2015. A Discussion of Qualifications and Skills in the Factory of the Future: A German and American Perspective. Technical report. VDI/ASME, (April 2015).

[19] Xavier Pi and Pere Tuset-Peiró. 2019. Los nuevos perfiles profesionales en el marco de la industria 4.0. *Oikonomics*, 12, (November 2019), 1–17. ISSN: 2339-9546. DOI: https://doi.org/10.7238/o.n12.1912.

[20] Benjamin Lindemann, Behrang Ashtari Talkhestani, Tobias Jung, Nada Sahlab, Nasser Jazdi, Wolfgang Schloegl, and Michael Weyrich. 2019. An architecture of an intelligent digital twin in a cyber-physical production system. *at - Automatisierungstechnik*, 67, 9, 762 –782. DOI: https://doi.org/10.1515/auto-2019-0039. https://www.degruyter.com/view/journals/auto/67/9/article-p762.xml.

[21] Sabrina Waffenschmidt. 2018. Paternity stablished. *Best Practice (T-Systems Magazine) - https://www.t-systems.com/en/best-practice/03-2018/experts/industry-4-0/digital-twin-840494.*

[22] Pieter van Schalkwyk. 2019. The Ultimate Guide to Digital Twins. Technical report. XMPro.

[23] E. Ries. 2011. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* Crown. ISBN: 9780307887917. https://books.google.es/books?id=tvfyz-4JIL wC.

[24] Steve Blank. 2003. *The Four Steps to the Epiphany: Successful Strategies for Products that Win.* Lulu Enterprises Incorporated. ISBN: 9781411601727. https://books.google.es/books?id=oLL2pjn2RV0C.

[25] Taylor Thompson. 2013. Building a minimum viable product? you're probably doing it wrong. *Harvard Business Review*. Retrieved 12/30/2017 from.

[26] Robert Sargent. 2011. Verification and validation of simulation models. In volume 37. (January 2011), 166 –183. DOI: 10.1109/WSC.2010.5679166.

[27] 2013. *Definition of virtual reality simulation models using specification and description language diagrams. SDL 2013: Model-Driven Dependability Engineering: 16th International SDL Forum, Montreal, Canada, June 26-28, 2013. Proceedings.* Springer Berlin Heidelberg, Berlin, Heidelberg, 258–274. ISBN: 978-3-642-38911-5.

[28] Pau Fonseca i Casas. 2006. Transforming sdl diagrams in a devs specification. In (September 2006). DOI: 10.13140/2.1.4973.9528.

[29] Hans Vangheluwe. 2000. Devs as a common denominator for multi-formalism hybrid systems modelling. *IEEE International Symposium on Computer-Aided Control System Design*, (August 2000). DOI: 10.1109/CACSD.2000.900199.

[30] Pau Fonseca i Casas. 2015. Transforming classic discrete event system specification models to specification and description language. *SIMULATION*, 91, 3, 249–264. DOI: 10.1177/0037549715571623. eprint: https://doi.org/10.1177/0037549715571623. https://doi.org/10.1177/0037549715571623.

[31] Pau Fonseca i Casas. 2009. Towards an automatic transformation from a devs to a sdl specification. In *Proceedings of the 2009 Summer Computer Simulation Conference* (SCSC '09). Society for Modeling & Simulation International, Istanbul, Turkey, 348–355.

[32] Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors. 1999. *Framework for automatic sdl to c++ translation. Formal Methods for Protocol Engineering and Distributed Systems: FORTE XII / PSTV XIX'99 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX) October 5–8, 1999, Beijing, China.* Springer US, Boston, MA, 95–115. ISBN: 978-0-387-35578-8. DOI: 10.1007/978-0-387-35578-8_6. https://doi.org/10.1007/978-0-387-35578-8_6.

[33] Xavier Pi. 2020. Sdl4snap! https://github.com/pixavier/sdl4snap. (2020).

[34] Edel Sherratt, Ileana Ober, Emmanuel Gaudin, Pau Fonseca I Casas, and Finn Kristoffersen. 2015. Sdl - the iot language. In *17th International System Design Languages Forum (SDL 2015).* Volume 9369. Thanks to Springer editor. This papers appears in Volume 9369 Lecture Notes in Computer Science ISSN : 0302-9743. ISBN: 978-3-319-24911-7 The original PDF is available at : http://link.springer.com/chapter/10.1007%2F978-3-319-24912-4_3. Springer-Verlag, Berlin, DE, 27–41. DOI: 10.1007/978-3-319-24912-4_3. http://oatao.univ-toulouse.fr/15419/.

[35] Donghang Guo, Eunice Santos, Landon Fraser, and Megan Olsen. 2005. A light-weight message transport framework for multi-agent based simulation. In (January 2005), 577–582.

[36] Günther Schuh, Reiner Anderl, Jürgen Gausemeier, Michael ten Hompel, and Wolfgang Wahlster, editors. 2017. *Industrie 4.0 Maturity Index: Managing the Digital Transformation of Companies. acatech Studie.* Utz, München. ISBN: 978-3-8316-4613-5.

[37] Digital Twin Consortium. 2020. Digital twin consortium charter. (2020). https://www.digitaltwinconsortium.org/pdf/DTC-Charter.pdf.

[38] E.D. Hirsch, J.F. Kett, and J.S. Trefil. 1988. *Cultural Literacy: What Every American Needs to Know. National bestseller.* Vintage Books. ISBN: 9780394758435. https://books.google.es/books?id=oaEveFmZgWIC.

[39] European Commission. 2020. Shaping Europe's Digital Future. Technical report. European Commission - European Union 2020.

[40] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Commun. ACM*, 52, 11, (November 2009), 60–67. ISSN: 0001-0782. DOI: 10.1145/1592761.1592779. https://doi.org/10.1145/1592761.1592779.

[41] Rina Diane Caballar. 2020. Programming without code: the rise of no-code software development. Online IEEE Spectrum. (March 2020). https://spectrum.ieee.org/tech-talk/computing/software/programming-without-code-no-code-software-development.

[42] Amy Groden-Morrison. 2019. Low-code and no-code development: manufacturers secret weapon for getting to industry 4.0. Online. (2019). https://www.alphasoftware.com/blog/low-code-and-no-code-development-manufacturers-secret-weapon-for-getting-to-industry-4.0.

[43] Paul Vincent, Kimihiko Iijima, Mark Driver, Jason Wong, and Yefim Natis. 2019. Magic Quadrant for Enterprise Low-Code Application Platforms. Technical report. Gartner. https://www.gartner.com/doc/reprints?id=1-1ODOM46A&ct=190812&st=sb.

[44] Eckart Modrow. 2018. *Computer Science with Snap!* emu-online Scheden.

[45] Xavier Pi. 2020. Mqtt4snap! https://github.com/pixavier/mqtt4snap. (2020).

[46] PragmaDev. 2018. *PragmaDev Studio Reference Manual*. PragmaDev.

[47] Ferenc Belina and Dieter Hogrefe. 1989. The ccitt-specification and description language sdl. *Computer Networks and ISDN Systems*, 16, 4, 311 –341. ISSN: 0169-7552. DOI: https://doi.org/10.1016/0169-7552(89)90078-0. http://www.sciencedirect.com/science/article/pii/0169755289900780.

[48] Pau Fonseca i Casas. 2010. 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, 419–426. http://portal.acm.org/citation.cfm?id=1999470\&CFID=29315481\&CFTOKEN=81718596.