Received October 26, 2021, accepted November 6, 2021, date of publication November 10, 2021,
date of current version November 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3127344

Describing Structure and Complex Interactions in Multi-Agent-Based Industrial Cyber-Physical Systems

LUIS RIBEIRO[1], (Senior Member, IEEE), AND LUIS GOMES[2], (Senior Member, IEEE)
[1]Division of Product Realization, Department of Management and Engineering, Linköping University, 581 83 Linköping, Sweden
[2]Center of Technology and Systems, Department of Electrical and Computer Engineering, NOVA University Lisbon, Lisbon, 2829-516 Costa da Caparica, Portugal

Corresponding author: Luis Ribeiro (luis.ribeiro@liu.se)

ABSTRACT The description of structure and complex interactions in Multi-agent-based Industrial Cyber-physical (MAS-ICPS) systems has been elusively addressed in the literature. Existing works, grounded on model-based engineering, have been successful at characterizing and solving system integration problems. However, they fail to describe accurately the collective and dynamic execution behaviour of large and complex industrial systems, particularly in more discrete production domains, such as: automotive, home appliances, aerospace, food and beverages, etc. In these domains, the execution flow diverts dynamically due to production disturbances, custom orders, fluctuations in demand in mixed model production, faults, quality-control and product rework, etc. These dynamic conditions require re-allocation and reconfiguration of production resources, redirection of production flows, re-scheduling of orders, etc. A meta-model for describing the structure and complex interactions in MAS-ICPS is defined in this paper. This contribution goes beyond the State-Of-The-Art (SOTA) as the proposed meta-model describes structure, as many other literature contributions, but also describes the execution behaviour of arbitrarily complex interactions. The previous is achieved with the introduction of general execution flow control operators in the meta-model. These operators cover, among other aspects, delegation of the execution flow and dynamic decision making. Additionally, the contribution also goes beyond the SOTA by including validation mechanisms for the models generated by the meta-model. Finally, the contribution adds to the current literature by providing a meta-model focusing on production execution and not just on describing the structural connectivity aspects of ICPSs.

INDEX TERMS Industrial cyber-physical systems, multiagent systems, complex interaction flows, modeling.

I. INTRODUCTION

Multiagent systems have historically been proposed, across a wide range of industrial application domains, as a solution for the creation, management and development of intelligent industrial systems [1]–[6]. However, despite the documented success cases, the adoption of agent concepts and technologies has been limited. While an in-depth discussion on the reasons for the elusive adoption is beyond the scope of this article, there is a consensus in the community that the lack of maturity of the supporting technologies is one of the main aspects [7], [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Weiguo Xia.

In the context previously described, the industrial automation domain, particularly at field level, is dominated by the programming languages described in the IEC61131-3 standard [9] and its progressive improvements which, recently, have come to include a form of object oriented programming. Simultaneously, the advent of Industry 4.0 is based in design principles that cannot be entirely satisfied by today's prevalent automation languages. The new design metaphors, Digital Twins (DTs) [10] and Industrial Cyber-Physical Systems seek to address the pressing need of producing sustainability while providing increasing personalized products.

Producing sustainability, under the mentioned circumstances, poses many challenges from a production execution perspective: increase of custom orders and less predictable ordering patterns, the need to produce different products in the same system while attaining high quality standards, the need to dynamically re-use production resources as a function of the lesser organized ordering patterns, being able to react quicker to systems faults and mitigate their impact in production either by rescheduling or re(allocating) resources or even by procuring and integrating temporarily resources from third parties, etc. Research in DTs and ICPSs has proven very fruitful in the development of models that characterize the structure of machines and systems that are statically interconnected. However, in quite many systems, relevant information is not semantically organized and will generally lack structure [11]. Contributions that define arbitrarily complex production execution flows have been relatively elusive in the literature. The authors concentrate essentially on discrete production systems and this contribution does not tackle the simulation or execution at the level of the physical continuous processes. In this sense, it is not a replacement for languages such as Modelica [12]. Furthermore, the authors briefly address DTs because quite many production system organizational models have been contributed under that design metaphor. Nevertheless, this paper will focus on ICPSs, without a loss of generalization of the contribution to DTs. It considers the ICPS scope discussed in [10] and [13] where the connection between the cyber dimension and the physical dimension is discussed and, hard real-time vs. soft-real time execution opportunities are modelled, respectively.

The known candidates for the implementation of Industrial Cyber-Physical Systems (ICPSs) are Service Oriented Architectures (SOA) [14], Multiagent-based Systems (MAS) and, most likely, a combination of both [15]. These have been in and out of the spotlight in the past two decades following different technological hypes and trends. ICPSs are seen as a specialization of the more generalized concept of Cyber-Physical System (CPS) [16]. ICPSs share the same challenges and problems of CPSs particularly the need to effectively orchestrate software and physical processes. This orchestration encompasses many computational levels [16] including the one where control transitions from an hard-real time process to a soft-real time one. It is at this interface that MAS and SOA may contribute the most. SOA advocate the use of semantically defined web-service endpoints, which are often stateless, and can be aggregated in multiple ways to create more complex services. MAS, on the other hand, provide the concepts and technologies for adaptive intelligent execution and the management of state information which is crucial in complex execution scenarios which may include negotiation and the harmonization of conflicting goals.

The need to process information in a more semantically harmonized way led several researchers to combine semantic and agent based technologies to address system modelling and integration, (self)reconfiguration and manufacturing execution problems. The use of semantic technologies solves many of these problems by providing the means to formally describe dictionaries and complex, non-obvious, relations between different entities in a system. It also allows the harmonization of data from multiple sources in a consistent and machine interpretable way. However, the description of complex interactions in a MAS, in industrial contexts, has not been adequately supported, particularly at the level of the cyber-physical interface. This challenge has been partially addressed, in discrete production systems, by recognizing that agent logic should be decoupled from the controller logic [17]. Some harmonization techniques have been developed between agents, programmed in JADE, and the IEC61499 [18], [19]. However, the definition of this interface remains largely ad hoc. The IEEE P2660.1 initiative [20] has collected and categorized a diversity of practices and introduced a scoring algorithm to access the suitability of a specific practice in different application contexts. The analysed practices do not provide, in a platform independent way, a generic model that relates the capabilities exposed by agents and their complex interactions, that often include several agents, to their low level execution.

This is one of the main challenges that this work seeks to address which is of significant importance for all the other works described in the literature that seek to address dynamic system reconfiguration in the scope of sustainable production, as discussed before. Most of them assume that the capabilities of cyber-physical resources can be described simply by the name of a function. This assumption, on the one hand, simplifies reconfiguration algorithms, on the other hand, is fairly unrealistic since industrial equipment functions require the definition of many parameters. The name alone cannot be used in capability matching. Another common assumption is that systems have a fixed functional granularity, connected to the previous simplification. This is clearly not the case for most modular industrial systems as it has been demonstrated that different levels of granularity impact important characteristics of the system such as: diagnosability, flexibility, re-configurability, running costs and maintenance, etc. [21].

While addressing the problem of describing complex interactions and articulating this agent-based execution consistently with field level execution, the proposed approach also creates the opportunity to describe the functions of a cyber-physical resource at an abstraction and with a granularity level useful for these other dynamic reconfiguration and optimization approaches. In this context, this paper focuses on providing a meta-model, in a platform and technology agnostic way, that addresses the identified challenges. In particular, it focuses on describing the system from an execution perspective using the concept of skill as the main executable element. In doing so, the meta-model does not prescribe or restrict any particular organization, search or matching strategies, but provides additional features that should be used to assess whether different skills are equivalent or may be combined. Such features include the parameters of the skills, and their types, and the expected execution flow of aggregated skills. These provide additional information that can be used for precisely locating, aggregating and executing functions beyond just their name.

The meta-model presented and discussed has been instantiated in concrete models and tested in a physical prototype using an experimental multiagent-based runtime environment, which is not detailed for the sake of brevity. Simultaneously, the model can be adapted to many different technologies and runtime environments. These potential adaptations do not alter its value and, on the contrary, attest its usefulness and inter-operable nature by allowing multiple implementation strategies, for example using service oriented technologies combined with agents. This is the main motivation for expressing the key details and directions of the meta-model in a platform agnostic format provided by UML.

In this previous context, the three main contributions of this work are:

- a new meta-model to describe arbitrarily complex interactions in Multi-Agent-based ICPSs;
- the introduction of validation mechanisms for the models created using the proposed meta-model;
- a meta-model that can combine structural system organizational aspects with behavioural ones.

The following details are organized as follows: Section II discusses related work and positions the current contribution in that context; Section III is the main contribution and describes the Light Mechatronic Agent Design Meta-Model (LMADE), discusses its requirements and rationale and describes the main components of the meta-model in UML; Section IV discusses a didactic modelling example; Section V discusses the methodology for validating the generated model as instance of the proposed meta-model: Section VI positions and discusses the main results and, finally, Section VII reflects on the main conclusions.

II. RELATED WORK

In recent years, the number of models and meta-models, either formalized as ontologies or other representations, for supporting different manufacturing activities has increased. Despite the different proposals, the great majority has not been distributed publicly, their industrial usage has been limited and the focus has been on characterizing machining or assembly, with an under-representation of process industries [22]. While a relatively high number of ontologies and descriptive models for production processes have focused on describing components, operations and materials included in the production process, the industrial agents community has been historically attempting to combine the more descriptive models, provided by ontologies, with manufacturing execution aspects. This paper is part of these latter efforts but focuses on describing complex interactions at the cyber-physical interface. Model-based engineering of industrial systems is a large and multidisciplinary field and different models will capture many different views of many different stakeholders. This section focuses therefore on works whose nature brings them closer to the ICPSs' problems/challenges space. In this context, the works mentioned, and compared to, focus on describing production processes' or product's characteristics, directly address execution aspects, or support system performance assessment and reconfiguration. Such dimensions and works have important contact points to the proposed contribution as detailed later. Many of the related works tend to follow or promote an agent-based system organization with strong similarities with the PROSA architecture [23], [24]. This means that the idea of products, resources and orders as active agents that manipulate the knowledge of the different models, or as agents that can be derived from the models themselves, is common to almost all the works mentioned. This is an important point because the proposed work also encourages such organization which is largely based on distributed, modular and intelligent cyber-physical components.

The contribution described in [25] proposes a method for generating, in a platform independent way, common 3D geometries in CAD tools. A similar approach is described in [26] where digital twins are used as a concept to capture and model product and process information in a machining context. The work described in [27] follows a similar approach but focuses on generating 3D printing instructions from a cloud environment. These contributions are relevant because, in their specific domains, they actively transform product data into execution data.

While the proposed contribution allows for something very similar, it differs, generally, from production domain specific contributions as [25]–[27] in that it does not address any specific domain. Instead, it provides a meta-model that can be used to model arbitrarily complex execution flows in any domain. The particular semantics of a domain can be included by using domain naming conventions on the elements of the model.

In [28] an ontology for automatic reconfiguration of a flexible mechatronic system is discussed. The ontology is updated and used by the system's agents to define the flow of operations, available resources, matching between operations, resources and controllers. This knowledge is then used to support execution and configuration by deploying and activating the appropriate system resources. Industrial controllers expose their fixed and pre-existing capabilities through IEC61499 function blocks that implement the configuration and execution protocols. A similar approach is considered in [29] where the focus was on generalizing the function blocks to allow safe runtime system reconfiguration.

In contrast to these previous works, this contribution differs from [28], [29] as it does not specifically address system reconfiguration but rather provides a model to describe complex interactions and, as such, can be used as a language upon which reconfiguration algorithms can be enacted.

Code generation has been considered in [30] where an extension of the GAIA methodology [31] for creating multiagent-based systems was used for design and generation of industrial-oriented agents. The proposed ontology allows the creation of an agent-based system composed by resource agents, and their capabilities, as well as order managers that coordinate the execution in respect to the available functionalities. The work in [32] also targets code generation through the modAt4rMS approach which is based on structural, execution and behavioural models of a Programmable Logic Controller (PLC)-based system and its functions, parameters and execution states. Tools for transforming system models into PLC function blocks, in the scope of Industry 4.0 applications using the RAMI4.0 reference architecture, have also been considered in [33] and [34]. Still in the scope of the RAMI4.0 reference architecture, the work detailed in [35] discusses a digital tool box to translate information across the different viewpoints of the RAMI4.0 architecture.

This contribution differs from [30]–[34] as it does not entail code generation but instead provides a specification that can be interpreted by an agent platform for the purpose of (re)configuring and executing complex interactions. The elements in the proposed model, describing operations and parameters of components of any given system, have some similarity with [32] but, in particular, the parameters are not restricted to PLC types.

In [36] a formal specification for Manufacturing Execution Systems (MES) is detailed. The model provides a structural and detailed overview of complex industrial PLC-based systems as collection of activities, elements, events, gateways, data objects and the relations between them. While [36] provides a language and representational elements more suitable for engineering PLC systems, the present work seeks to address agent-based architectures where PLCs may be used as supporting controllers but do not necessarily have to.

Models for agent-based local planning and deliberation are considered in [37], [38] and [39]. Both works provide comprehensive models for coordinating the execution between intelligent products and the available manufacturing resources. In [40] a model for characterizing products and their corresponding agent-based representation is discussed. The model defines the operations and materials included in the production process and the resulting products of these operations. Similar approaches are considered in [41], [42].

In comparison to [37]–[42], where deliberative processes are used to support execution, or the focus is on selecting appropriate operations but not necessarily on describing the execution details, the proposed work would act as a base language to describe these details or, at least, to harmonize the description of available system functions to a level where the parametrization complexity can be encapsulated.

Ontology-based event driven manufacturing strategies, for network of manufacturing units are discussed in [43] where an architecture to manage relevant production events is presented.

The proposed work differs from [43] since it does not target the interactions between different manufacturing facilities but rather supports the execution interactions within one facility and its components. Even if the proposed approach can model interactions between manufacturing facilities and its execution operators are event driven, the work in [43] focuses more explicitly in defining the producers and consumers of relevant events, while this work supports additional operations such as the definition of dynamic decisions and other less linear execution flows.

A visual-based notation for describing and visually analysing complex automation solutions is discussed in [44]. The proposed notation allows the definition of hard real-time constraints and requirements in PLC-based systems considering their hardware capabilities, from the model it is then possible to understand how to dimension the automation solutions and evaluate performance issues. The integration of this work with software aspects, and potentially agent oriented code, and its impact in performance assessment is considered in [45].

In contrast to [44] and [45], the proposed work does not seek to model performance aspects of the system, however it has been used as a basis for developing the work in [45] which may act as a complement of it. Furthermore, the proposed contribution also relies on a visual notation to describe both the meta-model and the models derived from it. In particular, it uses UML for this purpose as documented later.

An architecture to assess Multi-Agent-based ICPSs in respect to their reconfigurability potential is detailed in [46]. The proposed architecture explicitly relates the existence to the availability of functions in specific system resources. That architecture has been recently generalized to support any large and complex technical system using hetero-functional-graph theory for which a digital toolbox is available as described in [47].

In comparison to [46], [47] this work focuses on the definition of the execution of skills which is not tackled in [46], [47]. This work can therefore be seen as an implementation extension of the work defined in [46].

Overall, in the community developing Multi-agent-based Industrial Cyber-physical Systems, quite many contributions have been proposed for the structural modelling of ICPSs. However, much of this modelling has concentrated on organizational aspects or on the creation of domain/machine specific production instructions. This work differentiates itself by providing a mechanism for platform agnostic definition of systems with arbitrarily complex execution flows. It is not a replacement for the other efforts described but rather a complementary framework.

Overall this work differentiates itself by providing a mechanism for platform agnostic definition of systems with complex execution flows. It is not a replacement for the other efforts described but rather a complementary and missing tool.

III. LIGHT MECHATRONIC AGENT DESIGN META-MODEL

This section introduces the main contribution of this work which is the Light Mechatronic Agent Design meta-model (LMADE).

Before describing the technical details, it is worth mentioning that ICPS is a large and roughly defined domain. The number and nature of requirements that needs to be satisfied to support its desirable characteristics is unbounded.

A. LMADE REQUIREMENTS AND RATIONALE

Industrial requirements predate the emergence of CPSs, are linked to the need to create re-configurable industrial systems [48], [49] and include: universality, mobility, scalability, modularity, compatibility and, as importantly, the need to create metrics to evaluate system reconfigurability as a function of these characteristics.

Cyber-physical requirements reflect the need to articulate logical aspects of system organization and behaviour with its physical aspects. Cyber-physical adaptability is a central requirement for ICPSs [50]. Adaptability, however, is required at many levels, from the computational platforms and supporting programming and modelling languages [16], up to the cyber-physical formulation of industrial systems [51]. This paper concentrates on the latter which discusses ICPSs along the following dimensions: control path, granularity & modularity levels and aggregation strategy.

Within these previous dimensions the LMADE meta-model seeks to address the following requirements formulated in accordance with the structure and degree of obligation proposed in [52]. The LMADE meta-model shall:

- R1: describe, in an harmonized way, heterogeneous capabilities of industrial resources.
- R2: describe the aggregate capabilities of industrial resources.
- R3: describe the logical structure of the system as a function of its agents, their skills, and their deployment characteristics.
- R4: describe the logical execution behaviour of the system in respect to its skills while identifying the agents responsible for their execution statically or dynamically through negotiation processes.
- R5: not include assumptions about the application domain.
- R6: support the customization of structure and behaviour to specialized domains.
- R7: be specified in a language that allows its implementation using different technologies.
- R8: be possible to validate.

Previous developments in ICPS have produced contributions at different abstraction levels from high level design principles to hardware in the loop implementations [5], [46] with a predominance of the first where several well-known reference architectures have been subject to a few proof of concept implementations. While these reference architectures are sufficiently defined, their intermediate models leading to hardware in the loop prototypes have seldom been discussed. Requirements R1 - R8 address specifically these intermediate models.

R1 and R2 reflect the need to further generalize the description of industrial resources. Most reference architectures detail the main actors and their high level functions but rarely provide a language to describe the technical functions of the resources with sufficient detail and in a platform agnostic way. In other cases, these functions are described only by their name which has a limited semantic value.

R3 and R4 address the need to simultaneously describe the structure and behaviour of the system. Linked to R1 and R2 most architectures do not provide intermediate models for this and tend to focus in structural aspects. Furthermore, execution in an industrial context is frequently defined as a sequence of steps. However, as systems need to become more reactive to production or market disturbances, defining processes as simple sequences can be limiting. The skill concept used in LMADE allows the incorporation of dynamic decision making processes whose behaviour can be customized for different applications.

R5 addresses the need to not constrain the model to specific application domains. There is a predominance of ICPS models for manufacturing systems, however, it is possible to further generalize beyond manufacturing and enlarge the applicability scope to other ICPSs.

R6 recognizes that different application domains have non-generalizing characteristics and these must find their way into the model, without disrupting it.

R7 motivates the description of the meta-model in a way that is as technology agnostic as possible, so as to pave the way for it to be implemented in as many different technologies as possible.

Finally, R8 covers an inherent need of all models. The ability to validate structural and behavioural aspects of a system is of paramount importance for ICPSs.

The LMADE meta-model adheres to these requirements and is given using UML notations.

B. OVERVIEW

An LMADE model, as explained in greater detail henceforth, envisions a technical system, particularly an ICPS, as a collection of agents and their skills. An agent behaves as the cyber counterpart of a physical system, a process within it or both. When an agent abstracts a physical component/system it describes the functions that said system exposes to other agents (abstracting themselves other components/systems) and that may be used by these other agents collaboratively to control the execution of production in an ICPS. When an agent abstracts a process, it describes the dynamic interactions between other agents in the system. As mentioned before, an agent may also do both simultaneously, acting as an aggregator of processes and production resources, while acting as the cyber counter-part of a larger section of a technical system or even the whole system. Principles of Holonic organization as detailed in [53] and emanating from Arthur Koestler's definition of Holon apply directly.

Agents, in their different roles, will describe their functions as skills. A skill is what an agent knows how to do and details how it is done in the system, which other agents may be involved in the execution of the skill and which skills within these agents shall be used. Furthermore, a skill may be activated by sending the agent that owns the skill a recipe that details the values of the input parameters of the skill.

Collectively, agents and their skills completely describe the structure and behaviour of a multi-agent-based ICPS defined with the LMADE meta-model. The previous is captured through an hierarchical meta-model with the concept of *LmadeAgentSpecification* as a central component (Fig. 1). The LMADE meta-model and its instances are specified using UML without any relevant deviation from its norm. In the forthcoming text, complementary information about how to read the UML diagrams will be provided, however, readers are referred to the formal UML specification for the language details (`https://www.uml.org/`). Relevant meta-model concepts are described as UML classes, denoted by squared symbols where the name of the class occupies the top compartment of the symbol. Additional compartments may refer to properties of the concept, denoted by *attributes*, or functions of the concept, denoted by *operations*. Attributes have a name and a type. Operations have a name, parameters' list and may have a return value. The relations between the entities in the model are represented using the standard visual notations for describing different types of associations, their direction, nature and arity (this is denoted by the lines connecting the classes and the additional symbols related to them). Generally, throughout the meta-model definition, aggregations (lines with a white diamond) are used over composition associations (not represented in any of the figures discussed) to make very explicit the decoupled nature of the instances. The previous is intentional and promotes reusability. Indeed, many different agents may have the same skill, the skill does not cease to exist just because the agents do (the skill exists but is simply not available or instantiated), the same applies for all the identified aggregation associations. It is therefore possible to maintain a storage of LMADE concepts that can be cyber-physically instantiated upon convenience.

All the instances of concepts from the LMADE meta-model have a Global Unique Identifier (GUID) and a friendly name that are not represented in (Fig. 1). These attributes make every instance uniquely identifiable. While uniqueness is provided by the GUID, the friendly name provides a human readable name for the instance. Friendly names are case insensitive in LMADE. Unique identification of components in the meta-model's instances is important from a traceability point of view. Any change to a instances results in new unique identifier being assigned to the instance which, in turn, enables the tracking of the evolution of the model over time, as every evolution of an instance will have an uniquely identified ancestor, and ensures that different versions of the instances can co-exist but never be mistakenly confused during usage. This is a general characteristic of the meta-model that applies to all the instances of its components.

An *LmadeAgentSpecification* aggregates a set of skills (denoted by the directed line with the white diamond and the arity specification which describes that 1 agent has 0 or more skills). A *Skill* is an abstract class (denoted by the name of the class written in italic), with a set of specializations detailed later (Fig. 2), that models a capability that an agent provides to its surrounding environment. Abstract classes are not stand-alone concepts but rather contain, from a modelling

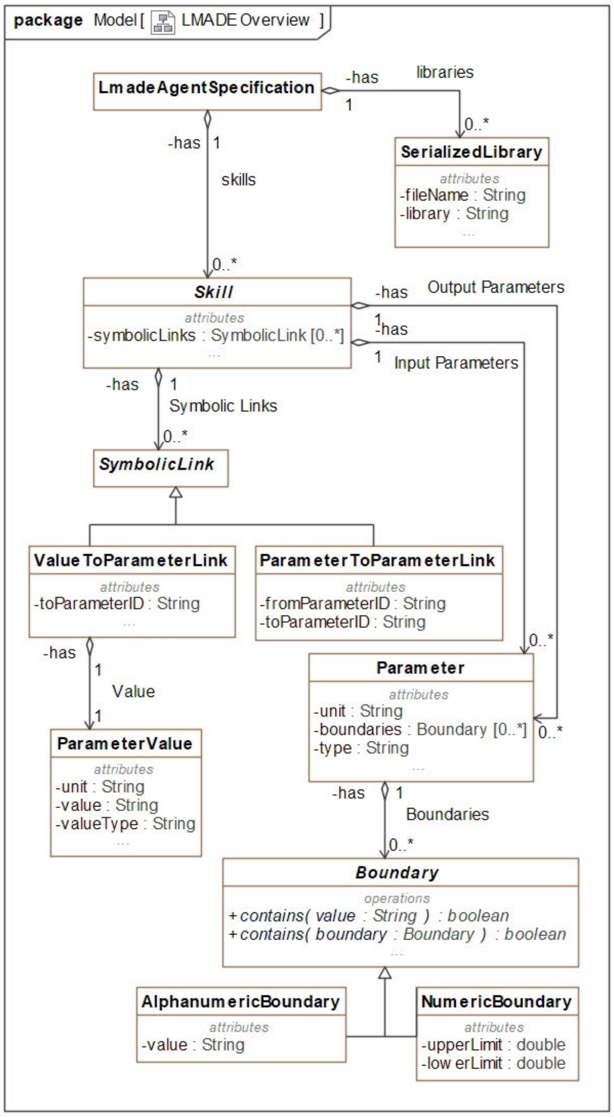


FIGURE 1. View of the LMADE meta-model focusing on parameterization of agents and skills.

perspective, a set of attributes and operations that are common to all their specializations. A specialized class is connected to a more generic class by a line with an white triangle. The concept of skill has been widely used in the agent-based literature and has become a *de facto* designation for the functions of an agent in an industrial context. However, its implementation has varied slightly to incorporate different requirements. In the LMADE context, a *Skill* has, zero or many, input and output parameters.

A *Parameter* is characterized by a *type*, which identifies its data type, and a *unit*, which identifies the magnitude of the quantity of the parameter. Parameters may have boundaries. A *Boundary* is a restriction to the admissible set of values of the parameter. It is also an abstract class that allows further extensions. The current implementation requires sub classes to implement two operations. The *contains* operation returns true if a particular value of a boundary or another boundary is contained in the set of possible values of an existing boundary. Currently, the model supports two specializations. The *NumericBoundary* allows the definition of numeric upper and lower values and the *AlphanumericBoundary* allows restricting string literals to specific values.

A *Skill* also aggregates symbolic links. A *SymbolicLink* defines a relation between the values of different parameters (*ParameterToParameterLink*) or between a particular value and a parameter (*ValueToParameterLink*). The purpose of the *ParameterToParameterLink* concept is to relate changes in the value of the originating parameter (*fromParameter*) to the value of the destination parameter (*toParameter*). A *ValueToParameterLink* asserts a similar relation but between a particular value, defined as a *ParameterValue*, and a destination parameter and can be used for initializing parameters. In runtime, symbolic links are used to copy and map values between parameters of the same or different skills. One may intuitively think of links as wires between parameters and values. Because any instance of the model, be it an agent or a parameter, has its own unique id a symbolic link can connect any pair of *ParameterValue* and *Parameter* or any pair of *Parameters*. A *ParameterValue* is defined by a *unit*, a *value* and a *valueType* that describe, respectively, the magnitude of the quantity of the parameter, the value, expressed in that quantity, and the data type of the value.

Finally, an *LmadeAgentSpecification* aggregates a set of serialized software libraries that are required to support its operation after it has been deployed. These libraries contain the domain specific implementations that specialize the behaviours of Atomic, Alternative and Loop skills, later detailed. While these libraries are mainly relevant in a runtime context they need to be attached to the instanciated model during the design phase. Since the agent definition aggregates all the skills whose behaviours may be customized for a specific domain, and some of these behaviours may use the same library, the supporting libraries are stored at agent level.

C. SKILL SPECIALIZATIONS

Fig. 2 provides additional details regarding the specification of skills in LMADE. The *Skill* abstract class contains six specializations: *AtomicSkill*, *AlternativeSkill*, *LoopSkill*, *SequentialSkill*, *SimultaneousSkill* and *DelegationSkill*. These specializations act as operators that control the flow of the execution of the skill as described next.

Skills also require a way of specifying an interface the physical world and for system specific execution semantics. For example, the *LoopSkill*, detailed next, will execute a skill cyclically until a certain condition is met. The condition is system specific and can be an arbitrarily complex decision making algorithm that needs to be harmonized with the LMADE model. A similar situation can be found in other skills, for example the *AtomicSkill* whose execution must activate native functions on the physical system its agent control. Here the translation between the LMADE execution semantics and the native system execution semantics is again system specific. These skills, that interface with native system/domain functions implement the *LibraryDeployer* interface which defines the minimal set of functions that a library class must implement to describe and locate its resources. This is the case of Atomic, Alternative and Loop skills that, as mentioned before, can be customized to different application domains. Their libraries are stored at agent level as *SerializedLibrary* class objects.

The *AtomicSkill* class models skills that integrate with external systems. Atomic skills traverse the cyber-physical barrier by integrating, for example, an agent with a physical asset. They may also integrate with other systems (other IT systems, libraries, algorithms, etc.). They implement the *LibraryDeployer* interface because they reference the low level libraries containing their implementations. These translate the execution of atomic skills (agent domain) into a particular action in the systems it interfaces with (physical/external system domain).

The *AlternativeSkill* class models an execution flow whereby one skill, among many alternatives, is scheduled for execution depending on a user defined condition. The list of possible alternatives eligible for selection and execution is represented by the *Alternatives*, one-to-one-or-many, aggregation. The user defined algorithm that performs the selection must implement the *AlternativeSkillDecisionMaker* interface that contains the *selectAlternative* operation. The implementation of the said operation must, given the set of the skill's input parameters' value and the list of the alternatives, return the skill selected for execution. The user defined implementation of the selection process requires the *AlternativeSkillDecisionMaker* to implement the *LibraryDeployer* interface.

The *LoopSkill* class models an execution flow whereby a skill runs in a loop until a given user defined condition is met. Similarly to the *AlternativeSkill*, the user has the opportunity to define the particular conditions that cause the loop to continue. This requires the implementation of the *LoopDecisionMakerInterface* and, in particular, the operation *continueLoop* which, based on the values of the input parameters of the skill, returns true if the loop should continue for a subsequent cycle or false otherwise.

The *SequentialSkill* class models an execution flow where all the sub-skills aggregated in the *Sequence* association are executed, one after the other, from the head to the tail of the list. The order should be strictly enforced and one skill must not start its execution before the previous has finished. The last skill must finish its execution for the sequential skill to finish its own.

The *SimultaneousSkill* class models an execution flow whereby all the skills aggregated in the *Simultaneous* association see their execution started at the same time. The simultaneous skill will follow-up the execution of these sub-skills and terminates its execution when all the sub-skills have finished their execution. All the sub-skills must therefore terminate, otherwise, the simultaneous skill does not terminate. The simultaneous skill does not enforce or preclude any kind of synchronization between the sub-skills. Furthermore,

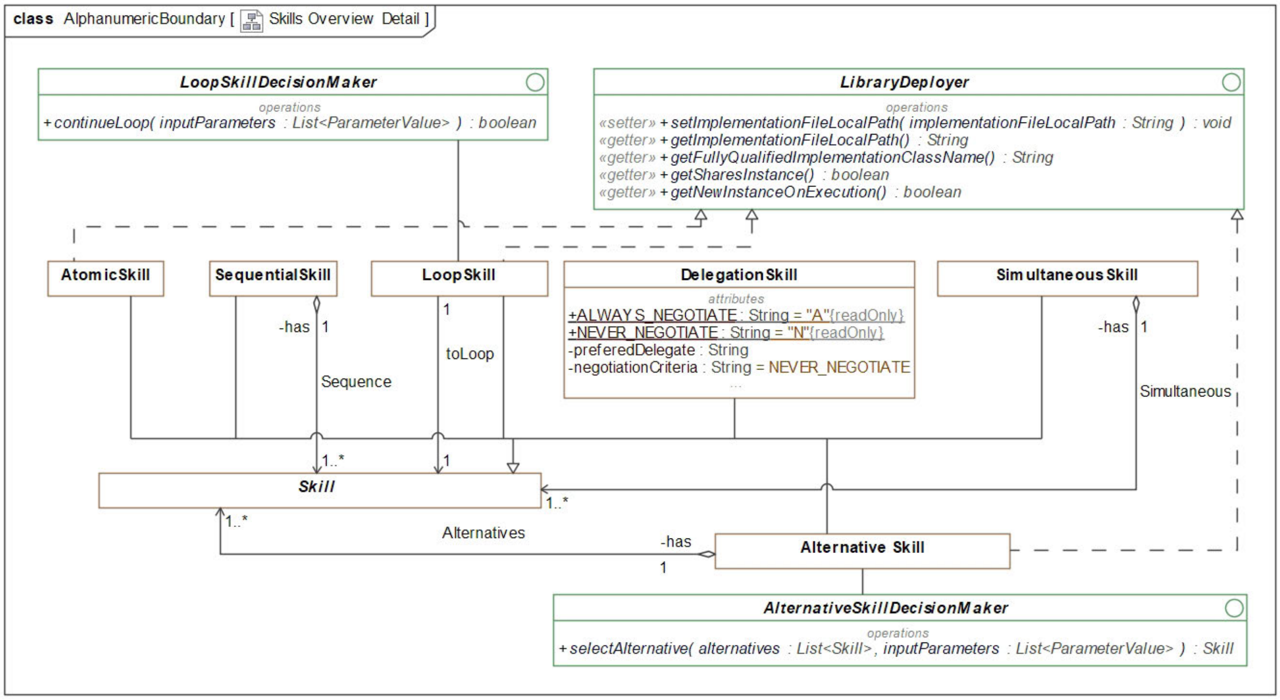

class AlphanumericBoundary [Skills Overview Detail]
LoopSkillDecisionMaker
operations
+continueLoop(inputParameters : List<ParameterValue>) : boolean
LibraryDeployer
operations
«setter» +setImplementationFileLocalPath(implementationFileLocalPath : String) : void
«getter» +getImplementationFileLocalPath() : String
«getter» +getFullyQualifiedImplementationClassName() : String
«getter» +getSharesInstance() : boolean
«getter» +getNewInstanceOnExecution() : boolean
AtomicSkill
SequentialSkill
LoopSkill
DelegationSkill
attributes
+ALWAYS_NEGOTIATE : String = "A"{readOnly}
+NEVER_NEGOTIATE : String = "N"{readOnly}
-preferedDelegate : String
-negotiationCriteria : String = NEVER_NEGOTIATE
SimultaneousSkill
-has 1
Sequence
1
toLoop
-has 1
Simultaneous
1..*
1
Skill
1..*
1..*
Alternatives
-has
1
Alternative Skill
AlternativeSkillDecisionMaker
operations
+selectAlternative(alternatives : List<Skill>, inputParameters : List<ParameterValue>) : Skill

FIGURE 2. View of the LMADE meta-model focusing on skills.

this skill does not prevent an agent from executing several skills simultaneously. Its purpose is to allow simultaneous execution within a specific skill aggregate and to guarantee that, within that scope only, the subsequent execution flow is not started before the simultaneous execution block terminates. Depending on the runtime implementation an agent may simultaneously execute an arbitrarily high number of skills but if these skills contain simultaneous skills within their flow, their execution follows the rules stated above.

Finally, the *DelegationSkill* class models an execution flow whereby one agent can delegate the execution of a skill on another agent. The agent to which the skill is delegated must contain a skill that is compatible with the first skill. Being compatible means that the name of the skill and its parameters are the same in the delegation and the delegated skill and that the parameters match in number and type. Furthermore if boundaries are specified, the ones of the skill where the execution is delegated must be more general or equal than the ones on the delegation skill. For example if the delegate skill specifies a numeric parameter restricted between 0 and 10, with all the other requirements fulfilled, the skill that carries out the execution must restrict that parameter on the same or on a larger interval. A -10 to 10 interval would work but a 5 to 10 would not.

This skill contains two private members that are relevant to control the delegation procedure:

- *preferedDelegate* - stores the name of the agent to whom this agent delegates the execution of the recipe.
- *negotiationCriteria* - stores a value that enables or disables a negotiation procedure to determine the agent to whom this agent delegates the execution of the recipe. The value of the negotiation criteria can only take one of the following constants defined in the class: *ALWAYS_NEGOTIATE*, *ALWAYS_NEGOTIATE_USE_PREFERED* and *NEVER_NEGOTIATE*.

The first defines unconditional negotiation and, if enabled, the value of the *preferedDelegate* should be ignored. The second enables negotiation in the unavailability of the *preferedDelegate*. The last enforces the use of the *preferedDelegate*.

D. SYSTEM AGGREGATION DEPLOYMENT AND EXECUTION

LMADE also provides support for aggregating and deploying whole systems and for specifying execution recipes (Fig. 3). As mentioned before, the LMADE meta-model enables the description of the behaviour of the system through the composition of agents and skills, collectively they "are" the system. A collection of agents and skills can be further aggregated by the concepts hereby specified. This is particularly important for deploying the system and bringing it into operation. The LMADE runtime environment, not described in this paper, contains specialized agents, called Deployment Agents (DAs), that are able to process the *SystemDeploymentSpecification* and *AgentDeploymentSpecification* described next. The DAs are therefore runtime entities that are active on the computational targets where the LMADE models can be deployed. Sending a *SystemDeploymentSpecification* to a DA will cause it to validate the specification and instantiate and deploy all the agents within it in its computational platform.

The *SystemDeploymentSpecification* class aggregates all the *AgentDeploymentSpecification* required to deploy a

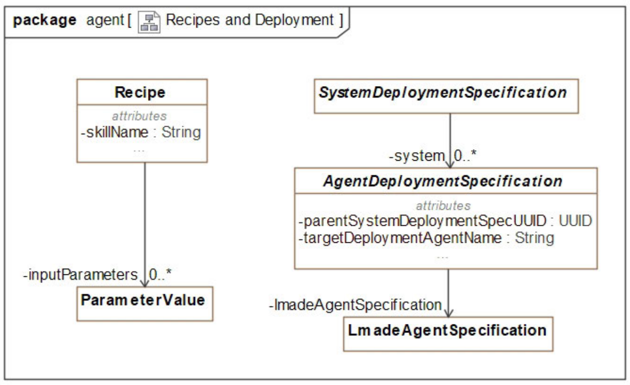

package agent [Recipes and Deployment]
Recipe
attributes
-skillName : String
-inputParameters 0..*
ParameterValue
SystemDeploymentSpecification
-system 0..*
AgentDeploymentSpecification
attributes
-parentSystemDeploymentSpecUUID : UUID
-targetDeploymentAgentName : String
-lmadeAgentSpecification
LmadeAgentSpecification

FIGURE 3. View of the LMADE model focusing on recipes and deployment.

particular system. The *AgentDeploymentSpecification* class requires the definition of two important attributes. The first is a reference to the parent system deployment specification, in the form of its unique id *parentSystemDeploymentSpecUUID*, and the second is the name of an agent, *targetDeploymentAgentName*. This name identifies the target agent that will be responsible for deploying the specified agent. This deployment agent is a runtime concept and should be able to receive the serialized version of the *AgentDeploymentSpecification* deserialize, process, and finally deploy it in an appropriate computational platform. Finally, the *AgentDeploymentSpecification* is associated to a particular *LmadeAgentSpecification* that fully describes an agent, its skills and supporting libraries to be deployed.

LMADE provides a model for the format used to send a request to an agent for the execution of a particular skill. The approach is to use recipes, a concept also widely explored in the specialized agent-based literature. A *Recipe* has a rather simple definition, akin to a remote procedure call. It contains the name of the skill to be executed and a set of *ParameterValue* that includes the concrete parametrization of the skill parameters. The recipe does not include any flow control modifiers or instructions. These are contained in the skill definitions themselves as explained before.

IV. WORKING EXAMPLE

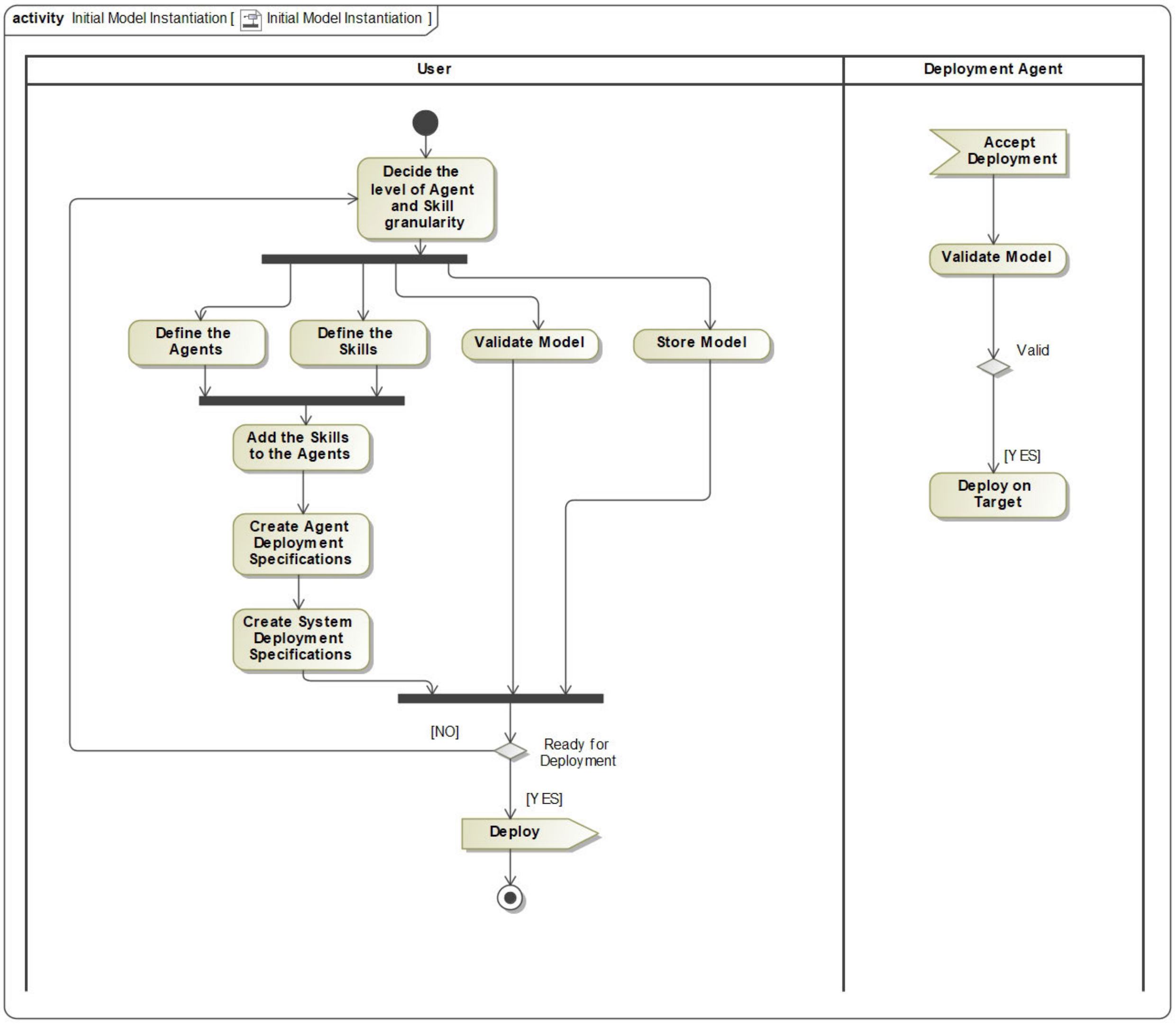
To illustrate the usage of a model instantiated from the LMADE meta-model one shall use the system in Fig. 5. The system consists of two rolling conveyor belts. The conveyors transfer parts between each other in a unidirectional way. If a conveyor is full, it should not accept parts from the other. Each conveyor contains a presence sensor that detects if a part is on it. This is not visible in the model and is abstracted by the conveyor's atomic skills.

The general model instantiation process is described in Fig. 4. The process starts with a decision on the level of granularity of agents and their skills. As discussed in [21] the level of service (in this case skill) granularity is a system specific decision that entails important trade-offs. In the example described next there was no specific technical criteria for the selected level of granularity, instead the example was devised to showcase the highest amount of entities from the LMADE meta-model. However, from an instantiation point of view regarding real systems this is a crucial step that needs to be carefully carried out. From that moment on, the user of the system, in charge of defining it, may start to define the agents and their skills. Because these elements are independent their development can happen in parallel and can involve many different users. At any step in the entire process any of the models can be validated and stored. With a first definition of agents and skills these need to be integrated (i.e. skills need to be attributed to specific agents). The previous is an iterative process and any number of skills may be added, removed or modified from existing agents. When an agent specification (*LMADEAgentSpecification*) is complete and is ready for deployment its agents and system deployment specifications can be created or modified. After these have been validated they may be sent to the deployment agent for instantiation and deployment on the target computational platform.

In the present case, each conveyor will be controlled and abstracted by an agent which constitutes the cyber-part of the cyber-physical aggregate. The discussed example has a didactic value only as mentioned before, it would not be the best technical solution for the problem at hand. However, it was designed to demonstrate, in a compact way, a large number of the features in the meta-model.

The instantiation of the conveyor agent is depicted in Fig. 6. The instantiation of the second conveyor is similar and omitted for brevity. In all the meta-model instantiation examples, in this section, the paper uses UML Object diagrams. Such diagrams are similar to the class diagrams earlier presented but, as the name suggests, provide a specific instantiation for the classe's attributes and operations. Each conveyor contains four skills: *startProductRelease*, *stopProductRelease*, *reserveIfFree* and *acceptProduct*. One takes advantage of this instantiation exercise to introduce some additional model parameters that have not been discussed in the model before, namely *sharesInstance* and *new instanceOnExecution* which apply to all the entities that are library deployers. The first refers to the ability of the skill, or the agent, to share an instance of the same library while executing if more than one agent, or skill, in the specification refer to the same library. The second refers to the need of creating a new instance of the library every time a skill or an agent is executed. Collectively, they allow the specification of the supporting libraries in different ways and may suit different execution and implementation needs.

In the present case, one assumes that there will be just one library instance controlling each conveyor and that the different skills are all defined in the same library. This makes sense since, at supporting library level, there may be the need to share global variables that capture the overall status of the conveyor. For that reason, the skills share the instance of the library. The library keeps track of its state and therefore should not be re-instantiated for every execution. Two of the skills have output parameters of Boolean type. In particular, the *reserveIfFree* skill will reserve the conveyor for a product



activity Initial Model Instantiation [Initial Model Instantiation]
User
Deployment Agent
Decide the level of Agent and Skill granularity
Define the Agents
Define the Skills
Validate Model
Store Model
Add the Skills to the Agents
Create Agent Deployment Specifications
Create System Deployment Specifications
[NO]
Ready for Deployment
[YES]
Deploy
Accept Deployment
Validate Model
Valid
[YES]
Deploy on Target

FIGURE 4. Instantiation procedure.

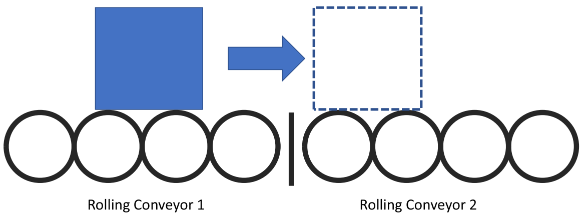


FIGURE 5. Example system.

handover procedure if the conveyor is free. If the reservation is successful the skill returns the value true in this output parameter. This means that the conveyor is now reserved and therefore not free Similarly, the *acceptProduct* skill, which is activated at the start of the handover procedure in the receiving conveyor to initiate its motion, returns true when the product has been completely handed over and terminates the handover procedure in the receiving end. The skill *startProductRelease* starts the motion and *stopProductRelease* stops the motion of the providing conveyor.

Note that the conveyors do not specify the actual handover procedure. This is a shared operation. In this scenario one chooses to model this operation as an additional skill on an additional agent called *HandoverAgent* (Figs. 7 and 8).

Figure 7 shows that the skill *handoverProductBetweenConveyors* is actually a complex flow of other skills. It is a sequence comprising a loop skill in step 0 (the first step in the execution of the sequential skill) and a simultaneous skill (defined in Fig. 8). Furthermore, Fig. 7 shows that the loop skill cyclically executes a delegation skill that delegates, in the agent Conveyor2, the execution of skill *reserveIfFree*. Additionally, the loop skill contains three input parameters two Boolean *control* and *reference* and one of type string *operator*. In this didactic example, the user specific implementation for the continuation of the loop (not represented in the figure) assumes that, for as long as the value of the control parameter differs from the value of the reference parameter, the loop continues. The user implementation will have access to the value of the parameters through the

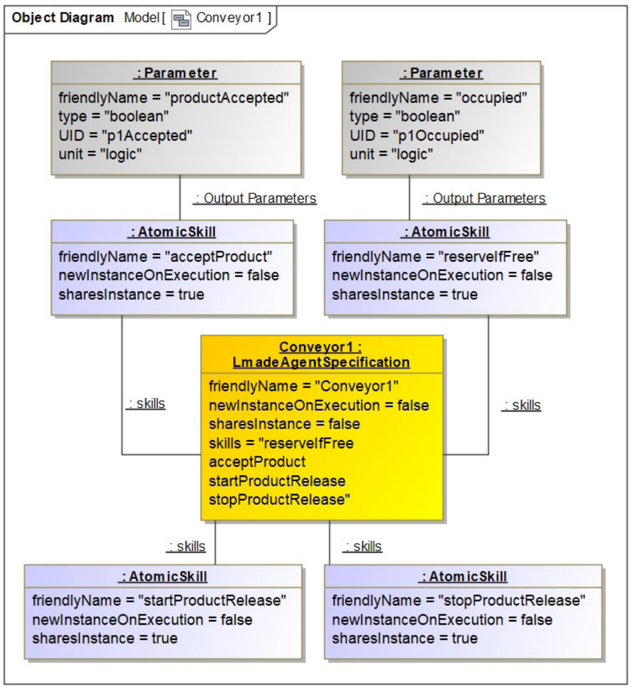

Object Diagram Model[Conveyor1]
: Parameter
friendlyName = "productAccepted"
type = "boolean"
UID = "p1Accepted"
unit = "logic"
: Parameter
friendlyName = "occupied"
type = "boolean"
UID = "p1Occupied"
unit = "logic"
: Output Parameters
: AtomicSkill
friendlyName = "acceptProduct"
newInstanceOnExecution = false
sharesInstance = true
friendlyName = "reserveIfFree"
Conveyor1 : LmadeAgentSpecification
friendlyName = "Conveyor1"
newInstanceOnExecution = false
sharesInstance = false
skills = "reserveIfFree acceptProduct startProductRelease stopProductRelease"
: skills
friendlyName = "startProductRelease"
friendlyName = "stopProductRelease"

FIGURE 6. Instantiation of the conveyor agent 1.

LoopSkillDecisionMaker interface earlier detailed (Fig. 2). This shows the possibility of customizing the skill with domain specific requirements. The inequality as a condition for the continuation is expressed using the corresponding value to parameter links that initialize the control parameter with the value *true*, and the reference and operator parameters with the values *false* and *!=* respectively. Finally, the parameter to parameter link establishing a dependency between the result of the delegated skill and the value of the control parameter ensures that, as soon as a reservation can be accepted by the second conveyor, the loop stops and the execution continues to step 1 (Fig. 8).

The continuation of the execution (Fig. 8) describes the simultaneous execution of a delegation skill and a sequential skill. The first delegates the *acceptProduct* skill in conveyor 2 while the second, first activates the release of the product, also by delegation in conveyor 1, and then uses a loop skill, in a pattern similar to what has been described before, to actively wait for the *productAccepted* to change its value to true. When the loop finishes, the handover is completed and the agent delegates the *stopProductRelease* skill in conveyor 1. This causes the overall handover procedure to terminate after all its skills have terminated their execution.

The instantiation of the Product Agent is now trivial and omitted for brevity. It requires the definition of an appropriate set of skills that would describe the production flow for that product. In the present case, it would associate the delegation of the handover skill to the specification of the product agent.

Finally, Fig. 9 shows an example of the deployment specification where, in the present case, a system deployment

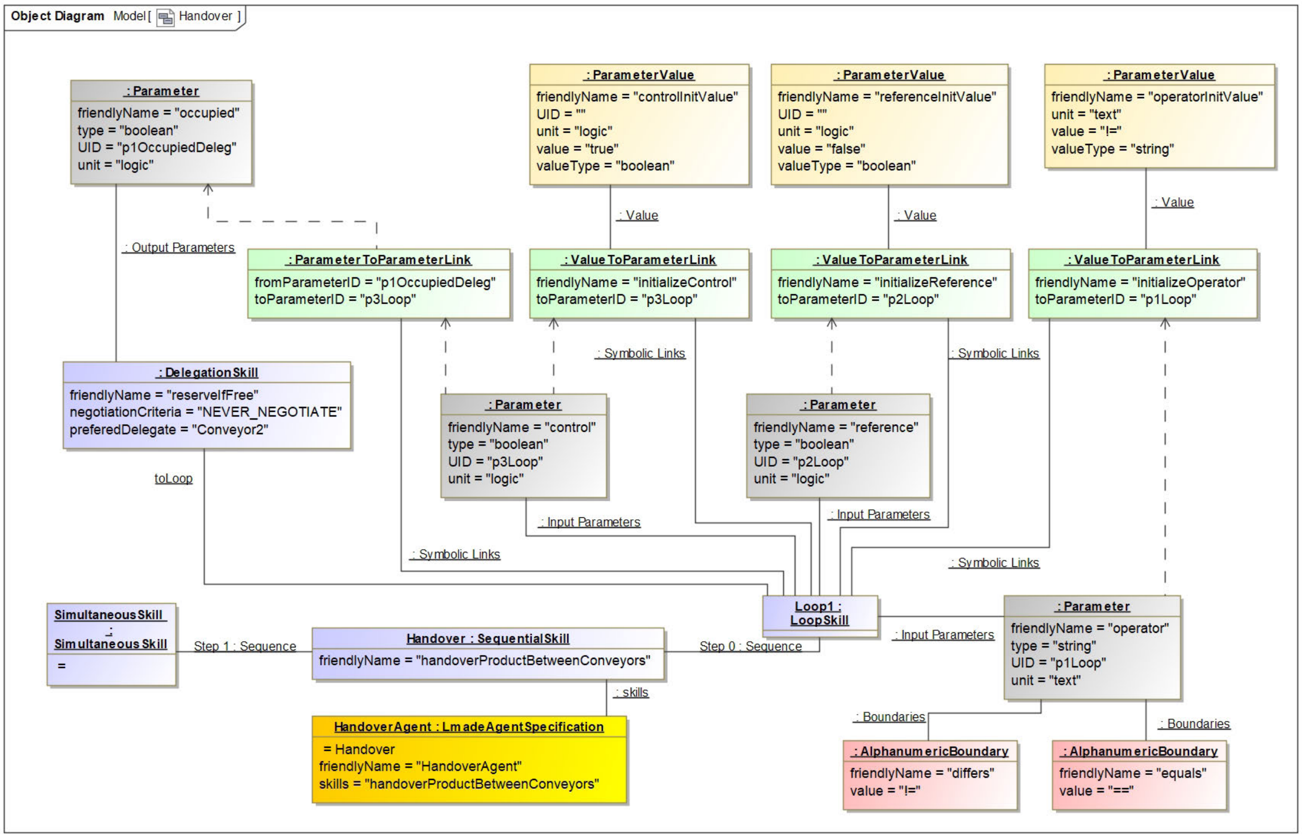

Object Diagram Model[Handover]
: Parameter
friendlyName = "occupied"
type = "boolean"
UID = "p1OccupiedDeleg"
unit = "logic"
: ParameterValue
friendlyName = "controlInitValue"
UID = ""
unit = "logic"
value = "true"
valueType = "boolean"
friendlyName = "referenceInitValue"
value = "false"
friendlyName = "operatorInitValue"
unit = "text"
value = "!="
valueType = "string"
: Value
: Output Parameters
: ParameterToParameterLink
fromParameterID = "p1OccupiedDeleg"
toParameterID = "p3Loop"
: ValueToParameterLink
friendlyName = "initializeControl"
toParameterID = "p3Loop"
friendlyName = "initializeReference"
toParameterID = "p2Loop"
friendlyName = "initializeOperator"
toParameterID = "p1Loop"
: Symbolic Links
: DelegationSkill
friendlyName = "reserveIfFree"
negotiationCriteria = "NEVER_NEGOTIATE"
preferedDelegate = "Conveyor2"
friendlyName = "control"
UID = "p3Loop"
friendlyName = "reference"
UID = "p2Loop"
toLoop
: Input Parameters
Loop1 : LoopSkill
SimultaneousSkill : Simultaneous Skill
=
Step 1 : Sequence
Handover : SequentialSkill
friendlyName = "handoverProductBetweenConveyors"
Step 0 : Sequence
friendlyName = "operator"
type = "string"
UID = "p1Loop"
unit = "text"
: skills
HandoverAgent : LmadeAgentSpecification
= Handover
friendlyName = "HandoverAgent"
skills = "handoverProductBetweenConveyors"
: Boundaries
: AlphanumericBoundary
friendlyName = "differs"
value = "!="
friendlyName = "equals"
value = "=="

FIGURE 7. Instantiation of the handover agent—part 1.

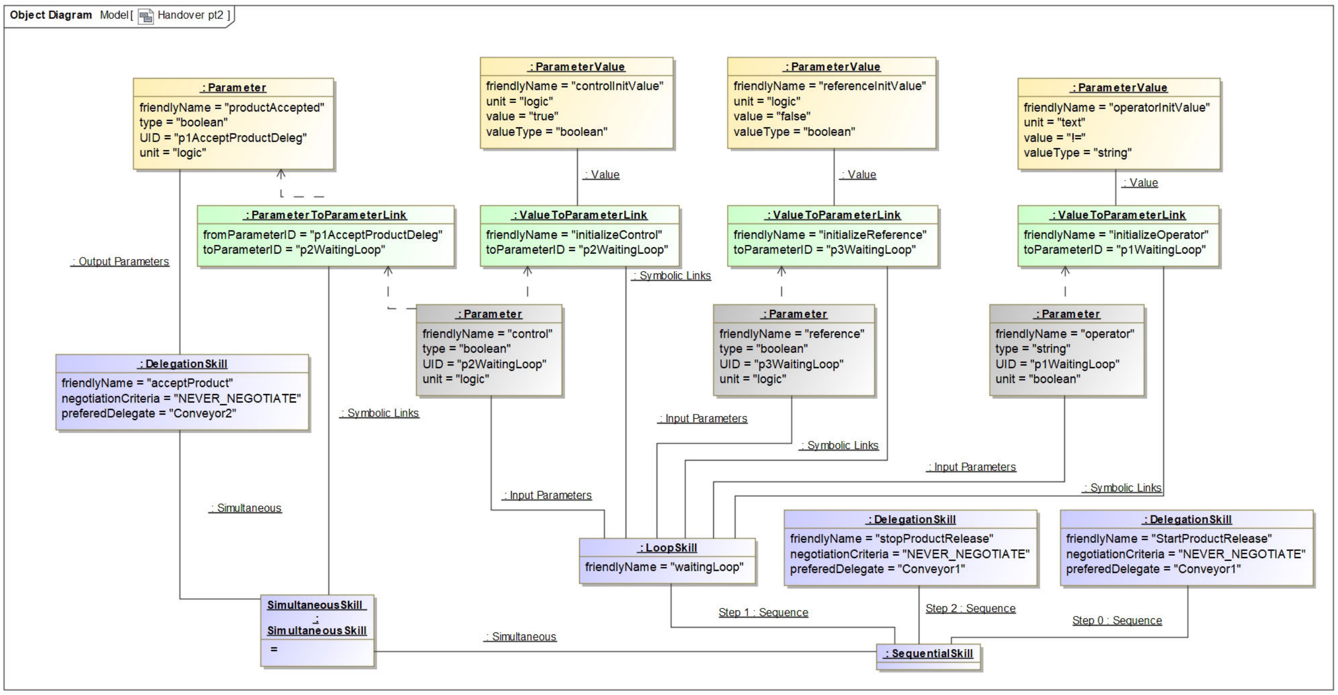

Object Diagram Model [Handover pt2]
: Parameter
friendlyName = "productAccepted"
type = "boolean"
UID = "p1AcceptProductDeleg"
unit = "logic"
: ParameterValue
friendlyName = "controlInitValue"
unit = "logic"
value = "true"
valueType = "boolean"
: ParameterValue
friendlyName = "referenceInitValue"
unit = "logic"
value = "false"
valueType = "boolean"
: ParameterValue
friendlyName = "operatorInitValue"
unit = "text"
value = "!="
valueType = "string"
: Value
: ParameterToParameterLink
fromParameterID = "p1AcceptProductDeleg"
toParameterID = "p2WaitingLoop"
: ValueToParameterLink
friendlyName = "initializeControl"
toParameterID = "p2WaitingLoop"
: ValueToParameterLink
friendlyName = "initializeReference"
toParameterID = "p3WaitingLoop"
: ValueToParameterLink
friendlyName = "initializeOperator"
toParameterID = "p1WaitingLoop"
: Output Parameters
: Symbolic Links
: Parameter
friendlyName = "control"
type = "boolean"
UID = "p2WaitingLoop"
unit = "logic"
: Parameter
friendlyName = "reference"
type = "boolean"
UID = "p3WaitingLoop"
unit = "logic"
: Parameter
friendlyName = "operator"
type = "string"
UID = "p1WaitingLoop"
unit = "boolean"
: DelegationSkill
friendlyName = "acceptProduct"
negotiationCriteria = "NEVER_NEGOTIATE"
preferedDelegate = "Conveyor2"
: Input Parameters
: Simultaneous
: LoopSkill
friendlyName = "waitingLoop"
: DelegationSkill
friendlyName = "stopProductRelease"
negotiationCriteria = "NEVER_NEGOTIATE"
preferedDelegate = "Conveyor1"
: DelegationSkill
friendlyName = "StartProductRelease"
negotiationCriteria = "NEVER_NEGOTIATE"
preferedDelegate = "Conveyor1"
SimultaneousSkill : SimultaneousSkill
=
Step 1 : Sequence
Step 2 : Sequence
Step 0 : Sequence
: SequentialSkill

FIGURE 8. **Instantiation of the handover agent—part 2.**

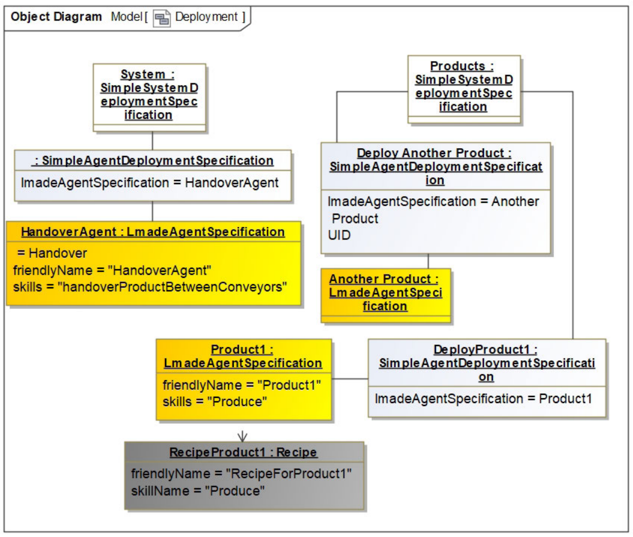

Object Diagram Model [Deployment]
System : SimpleSystemDeploymentSpecification
Products : SimpleSystemDeploymentSpecification
: SimpleAgentDeploymentSpecification
ImadeAgentSpecification = HandoverAgent
Deploy Another Product : SimpleAgentDeploymentSpecification
ImadeAgentSpecification = Another Product
UID
HandoverAgent : LmadeAgentSpecification
= Handover
friendlyName = "HandoverAgent"
skills = "handoverProductBetweenConveyors"
Another Product : LmadeAgentSpecification
Product1 : LmadeAgentSpecification
friendlyName = "Product1"
skills = "Produce"
DeployProduct1 : SimpleAgentDeploymentSpecification
ImadeAgentSpecification = Product1
RecipeProduct1 : Recipe
friendlyName = "RecipeForProduct1"
skillName = "Produce"

FIGURE 9. **Full system instantiation.**

specification would deploy the two conveyors (omitted for brevity) and the handover agent (depicted). Additional system deployment specifications would deploy an arbitrary number of products when needed, as well as additional resources being added to the system.

V. MODEL VALIDATION

Instances of the proposed model are validated using two procedures.

Firstly, Petri nets [54] can be used to validate the overall model of the multi-agent system, in terms of its main properties, namely completeness, deadlocks and livelocks freeness. For that, each of the six skill specializations is individually modeled. The Petri net model of each skill has one input place to allow the activation of the skill, and one output place denoting the completion of the activities associated with the skill, as illustrated in Fig. 10 b). Overall, the modeling of the whole system includes a source transition to activate the execution of the model, as well as a sinking transition to allow the confirmation of the conclusion of the execution, as presented in Fig. 10 a).

For a complete modeling of the system, namely of the parameters associated with the agents' execution and associated data transformations, high-level Petri nets are necessary to support full analysis of the system's properties. However, without loss of validity of the whole modeling strategy, low-level Petri nets are used in this paper for the modeling, still supporting completeness and liveness analysis.

The composition of individual skill models can be achieved through asynchronous composition obtained through fusion of input and output places [55], relying on the net addition operation, as proposed in [56].

Fig. 10 c) presents the model for *AtomicSkill*, where the transition models the completion of the skill. The *Sequential-Skill* results from the sequential composition of several skills through the fusion of the output place of one skill with the input place of the adjacent skill, as illustrated in Fig. 10 d) for a composition of a sequence of three skills. Fig. 10 d) right shows the resulting skill after fusion of input/output places of the individual skills. Fig. 10 e) presents the model for *AlternativeSkill*, where output places of the alternative skills and output place of the *AlternativeSkill* are fused together

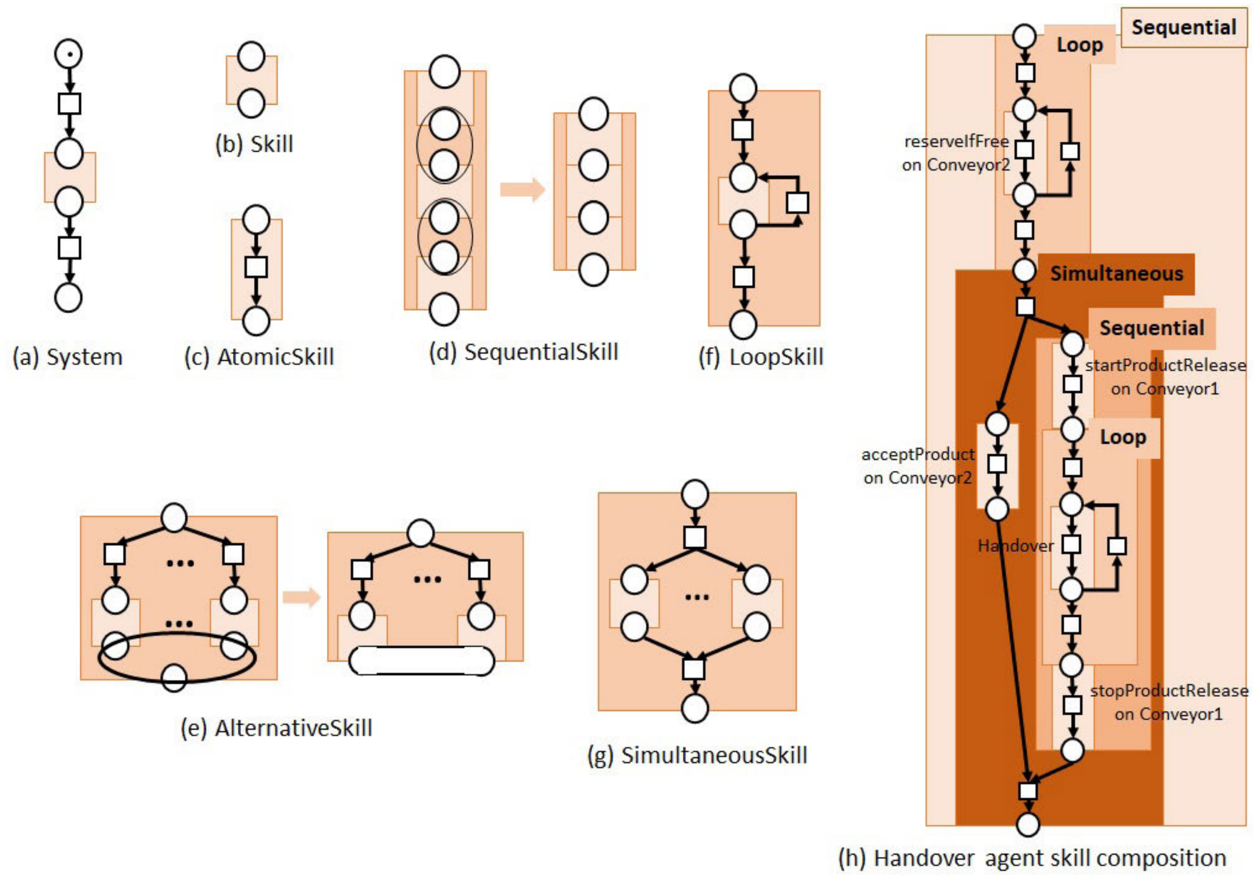


FIGURE 10. **Skills modeling and their composition.**

(using the net addition operation), as explicitly presented at Fig. 10 e). Fig. 10 f), and g) present the models for *LoopSkill*, and *SimultaneousSkill* (using fork and join constructs common in Petri net models), accordingly with behaviour descriptions previously presented in Section III-C.

The behavioral model for the handover agent of the working example is presented in Fig. 10 h), composed by a sequential composition of a *LoopSkill* waiting for availability of Conveyor 2 with a *SimultaneousSkill* having two concurrent skills, one to accept product on Conveyor 2, and other composed by the sequential composition of skills associated with Conveyor 1. Property verification of this model can be achieved building the associated reachability tree (the state-space associated with the execution of the Petri net model). Using the IOPT-Tools framework [57], it is concluded that the reachability tree has 19 states, and the model exhibits one deadlock associated with proper finalization of the execution.

Secondly, model instances are additionally validated recursively and, in addition to the conditions prescribed by the model itself in respect to the multiplicity of the associations, the following is checked for consistency. All the friendly names must be non-empty strings. The lower limit in numeric boundaries must be lower or equal to upper limit. The value of alphanumeric boundaries must be a non-empty string. Recipes must have a non-empty skill name. The type and unit of the parameters must have a value. The same happens for parameter values where, additionally, the value must also be non-empty. If a parameter value contains a type that is numeric then the value must also be valid within that type. Symbolic links may not have null values on their parameters. Parameter to parameter links are only valid if they connect two existing parameters with the same type and compatible constraints. Value to parameter links are only valid if the parameter value and the parameter they connect follow similar rules. Alternative, sequential, simultaneous and loop skills must contain sub-skills, as expressed by the 1 to 1 or many directed associations in Fig. 2. Serialized libraries must have non-empty filenames and libraries. Agent deployment specifications must contain non-empty target deployment agent names and agent specifications.

Collectively, these two validation procedures attest the fitness of the model for runtime deployment and execution.

VI. RESULTS AND DISCUSSION

The paper now address the three main contribution of the work as defined in the introduction:

1) a new meta-model to describe arbitrarily complex interactions in Multi-Agent-based ICPSs;
2) the introduction of validation mechanisms for the models created using the proposed meta-model;

3) a meta model that can combine structural system organizational aspects with behavioural ones.

Multi-agent-based system are, as a concept and technology, one of the most promising media for the implementation of the next generation of industrial automation solutions. Such systems have be subject to intensive research in the past. However, the existing models fall short of systematically describing complex execution flows in large industrial systems in a scalable way.

In respect to contribution 1), The proposed meta-model contributes to the body of knowledge by providing the concepts required to describe arbitrarily complex interactions in systems with multiple autonomous entities. It does so in a language that is independent from the system but that can be customized to fit system specific semantics. Flow control skills supporting cycles, dynamic decision, parallel or sequential execution can be used to describe the flow of execution of virtually any engineered system with more or less granularity. At the same time, the ability to be able to name skills and their parameters, units and constraints, according to system specific semantics and to be able to redirect system specific decision-making to particular decision making algorithms that make use of that semantic information ensures that, using the LMADE meta-model, all generalizable execution aspects are harmonized in a transparent way with system specific execution.

The previous is shown in the didactic example discussed in Section IV where the different execution flows are described as different skills from the meta-model and their names and units use system specific semantics. Non-generalizable execution is deferred by the atomic skills to the corresponding implementation libraries. The same happens to the loop skills that rely on system specific semantics to break or continue the loop. Because the outcome of the deferred executions either succeeds of fails it is possible to analyse the behaviour of the system in a system independent way.

To the best of the authors knowledge no other agent-based meta-model has openly provided such functionality.

The second contribution is a direct consequence of the first. Because the meta-model allows the definition and generalization of arbitrarily complex execution flows, it then becomes possible to use validation mechanisms on the models. Validation is of extreme value here because it attests the correctness of the execution flows. The discussed validation mechanisms cover both syntactic and semantic validation aspects. For example, a skill model is only valid if any set of connect parameters used the same units. Simultaneously, a skill is also only valid if the execution flow it describes has a start and an end, while not introducing unintended deadlocks.

This is detailed in Section V where the process of analysing a model using Petri nets is described and the discussed example is analysed. Furthermore, the LMADE meta-model generates models whose structure can be recursively parsed and validated, with the advantage that different sections of the model can be individually and locally validated and then potentially re-used in other models and subjected to a wider validation.

Given the state-of-the-art literature in the domain in which we position our work, the authors believe this is an important contribution to the body of knowledge.

Finally, the third contribution results from the meta-model seamlessly articulating the system structure with its behaviour. This is a key aspect for cyber-physical system modelling as understood in the context of this paper. The existing literature includes proposals for structural and behavioural meta-models. However, meta-models combining both, in addition to the two previous contributions are extremely elusive in the available literature.

VII. CONCLUSION AND FUTURE WORK

This work provides a meta-model for defining and associating the structure of a multi-agent-based cyber-physical industrial system with its complex interactions. The existing literature has been somehow elusive in supporting such models. Works describing the structure of systems generally fall short of modelling how that structure relates to the many complex interaction/execution flows that are required to run these systems. Simultaneously, quite many works try to optimize the usage of system resources by planning their execution. However, most of these works assume a unrealistically simple representation of the capabilities and functions of the cyber-physical resources in an industrial context.

The focus of the LMADE meta-model is to provide a framework for describing arbitrarily complex interaction in a generalizable way, while deferring system specific execution to appropriate system specific handlers. This allows models to be comparable and validated. It also allows models to be seamless integrated with each other and validated in different scopes and contexts.

Collectively, all the facets that the meta-model transposes to the models it generates make it quite useful in filling in the gap between the high heterogeneity of physical industrial resources and the much needed harmonization required for enacting intelligent and autonomous decision-making processes.

The present meta-model is not without limitations. By its nature it applies generally to Industrial Cyber-Physical Production Systems that operate on discrete production systems. Its application to continuous processes requires a much more careful assessment of the design of the skills and the agents. The skills, in particular, must entail functions that have a clear start and finish cycle, or that otherwise generate events that can be easily integrated into the skills. In its current version, the execution environment for the models also follows this principle and is event-driven.

On discrete production systems the meta-model and the models it generates apply directly with very little limitations. The most obvious limitation is the ability to follow-through hard-real-time processes. This is due to the nature of the skills controlling the flow of execution and the need to defer certain execution moments to system specific libraries. While the skill structure could, in principle, execute in a predictable time, from the moment the execution is deferred to a system specific library, the meta-model does not have the tools to predict the outcome of such execution in respect to its success or timing.

In its current form the LMADE meta-model has been successfully demonstrated in a few industrial systems at a prototype level. Future work will focus on the automatic translation of the model in to Petri net frameworks for an instantaneous validation of the execution flows and small improvements to the meta-model in respect to defining parameter boundaries. The latter will go in the direction of allowing better extension mechanisms that enable setting constraints on parameters with non-native types.

REFERENCES

[1] M. Wooldridge, *An Introduction to Multiagent Systems*. Hoboken, NJ, USA: Wiley, 2009.

[2] S. Bussmann, N. R. Jennings, and M. Wooldridge, *Multiagent Systems for Manufacturing Control: A Design Methodology*. Berlin, Germany: Springer-Verlag, 2013.

[3] L. Monostori, J. Váncza, and S. R. T. Kumara, "Agent-based systems for manufacturing," *CIRP Ann. Technol.*, vol. 55, no. 2, pp. 697–720, 2006.

[4] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie, "Applications of agent-based systems in intelligent manufacturing: An updated review," *Adv. Eng. Informat.*, vol. 20, no. 4, pp. 415–431, Oct. 2006.

[5] P. Leitão, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart agents in industrial cyber–physical systems," *Proc. IEEE*, vol. 104, no. 5, pp. 1086–1101, May 2016.

[6] P. Leitao, V. Marik, and P. Vrba, "Past, present, and future of industrial agent applications," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2360–2372, Nov. 2013.

[7] V. Marik and D. McFarlane, "Industrial adoption of agent-based technologies," *IEEE Intell. Syst.*, vol. 20, no. 1, pp. 27–35, Jan. 2005.

[8] S. Karnouskos, P. Leitao, L. Ribeiro, and A. W. Colombo, "Industrial agents as a key enabler for realizing industrial cyber-physical systems: Multiagent systems entering industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 14, no. 3, pp. 18–32, May 2020, doi: 10.1109/MIE.2019.2962225.

[9] M. Tiegelkamp and K.-H. John, *EC 61131-3: Programming Industrial Automation Systems*, vol. 14. Heidelberg, Germany: Springer, 1995.

[10] Y. Lu, C. Liu, K. I.-K. Wang, H. Huang, and X. Xu, "Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues," *Robot. Comput.-Integr. Manuf.*, vol. 61, Feb. 2020, Art. no. 101837.

[11] K. Nagorny, S. Scholze, A. W. Colombo, and J. B. Oliveira, "A DIN spec 91345 RAMI 4.0 compliant data pipelining model: An approach to support data understanding and data acquisition in smart manufacturing environments," *IEEE Access*, vol. 8, pp. 223114–223129, 2020.

[12] *Modelica—A Unified Object-Oriented Languagefor Systems Modeling—Language Specification Version 3.5*. Accessed: Jul. 1, 2021. [Online]. Available: https://modelica.org/documents/MLS.pdf

[13] L. Ribeiro and M. Hochwallner, "Time-related constraints in administration shell design within cyber-physical production systems," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2019, pp. 1564–1569.

[14] A. W. Colombo, "Industrial cloud-based cyber-physical systems," in *The IMC-AESOP Approach*, vol. 22. Cham, Switzerland: Springer, 2014, pp. 4–5.

[15] L. Ribeiro, J. Barata, and P. Mendes, "MAS and SOA: Complementary automation paradigms," in *Proc. Int. Conf. Inf. Technol. Balanced Autom. Syst.* Boston, MA, USA: Springer, 2008, pp. 259–268.

[16] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11st IEEE Int. Symp. Object Component-Oriented Real-Time Distrib. Comput. (ISORC)*, May 2008, pp. 363–369.

[17] P. Vrba, P. Tichý, V. Maík, K. H. Hall, R. J. Staron, F. P. Maturana, and P. Kadera, "Rockwell Automation's holonic and multiagent control systems compendium," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 1, pp. 14–30, Jan. 2011.

[18] O. J. L. Orozco and J. L. M. Lastra, "A real-time interface for agent-based control," in *Proc. Int. Symp. Ind. Embedded Syst.*, 2007, pp. 49–54.

[19] I. Hegny, O. Hummer, A. Zoitl, G. Koppensteiner, and M. Merdan, "Integrating software agents and IEC 61499 realtime control for reconfigurable distributed manufacturing systems," in *Proc. Int. Symp. Ind. Embedded Syst.*, Jun. 2008, pp. 249–252.

[20] *IEEE P2660.1—Recommended Practices on Indrmation Agents: Integration of Software Agents and Low Level Automation Functions*. Accessed: Sep. 2021. [Online]. Available: https://standards.ieee.org/develop/project/2660.1.html

[21] A. Homay, M. de Sousa, A. Zoitl, and M. Wollschlaeger, "Service granularity in industrial automation and control systems," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 132–139.

[22] L. Ramos, "Semantic web for manufacturing, trends and open issues," *Comput. Ind. Eng.*, vol. 90, pp. 444–460, Dec. 2015.

[23] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: PROSA," *Comput. Ind.*, vol. 37, no. 3, pp. 255–274, Nov. 1998.

[24] P. Valckenaers, "Perspective on holonic manufacturing systems: PROSA becomes ARTI," *Comput. Ind.*, vol. 120, Sep. 2020, Art. no. 103226.

[25] T. A. Tran and A. Lobov, "Ontology-based model generation to support customizable KBE frameworks," *Proc. Manuf.*, vol. 51, pp. 1021–1026, Dec. 2020.

[26] B. Caesar, A. Hanel, E. Wenkler, C. Corinth, S. Ihlenfeldt, and A. Fay, "Information model of a digital process twin for machining processes," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2020, pp. 1765–1772.

[27] W. Lepuschitz, T. Trautner, M. Mayerhofer, S. M. Fallah, I. Ayatollahi, and M. Merdan, "Applying ontologies in a cloud manufacturing system," in *Proc. 45th Annu. Conf. Ind. Electron. SoC.*, Oct. 2019, pp. 2928–2933.

[28] Y. Alsafi and V. Vyatkin, "Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing," *Robot. Comput.-Integr. Manuf.*, vol. 26, no. 4, pp. 381–391, Aug. 2010.

[29] W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan, "Toward self-reconfiguration of manufacturing systems using automation agents," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 1, pp. 52–69, Jan. 2011.

[30] C. Alexakos, M. Georgoudakis, A. P. Kalogeras, and S. Likothanassis, "Adaptive manufacturing utilizing ontology-driven multi-agent systems: Extending Pabadis' Promise approach," in *Proc. IEEE Int. Conf. Ind. Technol.*, Mar. 2012, pp. 42–47.

[31] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design," *Auton. Agents Multi-Agent Syst.*, vol. 3, no. 3, pp. 285–312, 2000.

[32] M. Obermeier, S. Braun, and B. Vogel-Heuser, "A model-driven approach on object-oriented PLC programming for manufacturing systems with regard to usability," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 790–800, Jun. 2015.

[33] J. Zhang, B. Ahmad, R. Harrison, A. W. Colombo, and S. Raasch, "An approach for resource function block generation: Towards rami 4. 0-compliant PLC programming," in *Proc. IEEE 18th Int. Conf. Ind. Informat. (INDIN)*, vol. 1, Dec. 2020, pp. 595–600.

[34] U. Gangoiti, A. López, A. Armentia, E. Estévez, and M. Marcos, "Model-driven design and development of flexible automated production control configurations for industry 4.0," *Appl. Sci.*, vol. 11, no. 5, p. 2319, Mar. 2021.

[35] C. Binder, C. Neureiter, and A. Lüder, "Towards a domain-specific approach enabling tool-supported model-based systems engineering of complex industrial Internet-of-Things applications," *Systems*, vol. 9, no. 2, p. 21, Mar. 2021.

[36] M. Witsch and B. Vogel-Heuser, "Towards a formal specification framework for manufacturing execution systems," *IEEE Trans. Ind. Informat.*, vol. 8, no. 2, pp. 311–320, May 2012.

[37] S. Rehberger, L. Spreiter, and B. Vogel-Heuser, "An agent approach to flexible automated production systems based on discrete and continuous reasoning," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2016, pp. 1249–1256.

[38] I. Kovalenko, K. Barton, and D. Tilbury, "Design and implementation of an intelligent product agent architecture in manufacturing systems," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[39] I. Kovalenko, D. Tilbury, and K. Barton, "The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems," *Control Eng. Pract.*, vol. 86, pp. 105–117, May 2019.

[40] M. López, J. Martín, U. Gangoiti, A. Armentia, E. Estévez, and M. Marcos, "Supporting product oriented manufacturing: A model driven and agent based approach," in *Proc. IEEE 16th Int. Conf. Ind. Inform. (INDIN)*, Jun. 2018, pp. 133–139.

[41] P. Vrba, M. Radakoviá, M. Obitko, and V. Maík, "Semantic technologies: Latest advances in agent-based manufacturing control systems," *Int. J. Prod. Res.*, vol. 49, no. 5, pp. 1483–1496, Mar. 2011.

[42] J. Dias-Ferreira, L. Ribeiro, H. Akillioglu, P. Neves, and M. Onori, "BIOSOARM: A bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors," *J. Intell. Manuf.*, vol. 29, no. 7, pp. 1659–1682, Oct. 2018.

[43] H. Cao, X. Yang, and R. Deng, "Ontology-based holonic event-driven architecture for autonomous networked manufacturing systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 1, pp. 205–215, Jan. 2021.

[44] E. Trunzer, A. Wullenweber, and B. Vogel-Heuser, "Graphical modeling notation for data collection and analysis architectures in cyber-physical systems of systems," *J. Ind. Inf. Integr.*, vol. 19, Sep. 2020, Art. no. 100155.

[45] D. Hujo, B. Vogel-Heuser, and L. Ribeiro, "Towards a graphical modelling tool for response-time requirements based on soft and hard real-time capabilities in industrial cyber-physical systems," *IEEE J. Emerg. Sel. Topics Ind. Electron.*, early access, Jun. 29, 2021, doi: 10.1109/JESTIE.2021.3093248.

[46] A. M. Farid and L. Ribeiro, "An axiomatic design of a multiagent reconfigurable mechatronic system architecture," *IEEE Trans. Ind. Informat.*, vol. 11, no. 5, pp. 1142–1155, Oct. 2015.

[47] D. Thompson, P. Hegde, W. C. H. Schoonenberg, I. Khayal, and A. M. Farid, "The hetero-functional graph theory toolbox," 2020, *arXiv:2005.10006*.

[48] H. A. El Maraghy, "Flexible and reconfigurable manufacturing systems paradigms," *Flexible Service Manuf. J.*, vol. 17, no. 4, pp. 261–276, 2006.

[49] A. M. Farid, "Measures of reconfigurability and its key characteristics in intelligent manufacturing systems," *J. Intell. Manuf.*, vol. 28, no. 2, pp. 353–369, Feb. 2017.

[50] L. Ribeiro and M. Hochwallner, "On the design complexity of cyberphysical production systems," *Complexity*, vol. 2018, Jun. 2018, Art. no. 4632195.

[51] L. Ribeiro and M. Björkman, "Transitioning from standard automation solutions to cyber-physical production systems: An assessment of critical conceptual and technical challenges," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3816–3827, Dec. 2018.

[52] C. Rupp, *Requirements-Engineering Und-Management: Professionelle, Iterative Anforderungsanalyse Praxis*. München, Germany: Hanser-Verlag, 2007.

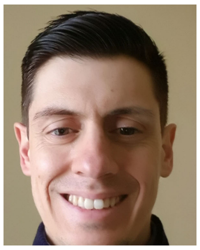
[53] P. Valckenaers and H. Van Brussel, *Design for the Unexpected: From Holonic Manufacturing Systems Towards a Humane Mechatronics Society*. Oxford, U.K.: Butterworth-Heinemann, 2015.

[54] C. Girault and R. Valk, *Petri Nets for Systems Engineering—A Guide to Modeling, Verification, and Applications*. Berlin, Germany: Springer-Verlag, 2003.

[55] L. Gomes and J. P. Barros, "Structuring and composability issues in Petri nets modeling," *IEEE Trans. Ind. Informat.*, vol. 1, no. 2, pp. 112–123, May 2005.

[56] J. P. Barros and L. Gomes, "Net model composition and modification by net operations: A pragmatic approach," in *Proc. 2nd IEEE Int. Conf. Ind. Inform.*, Jun. 2004, pp. 309–314.

[57] L. Gomes, F. Moutinho, and F. Pereira, "IOPT-tools—A web based tool framework for embedded systems controller development using Petri nets," in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–6.



LUIS RIBEIRO (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical and computer engineering, with specialization in robotics and computer integrated manufacturing, from NOVA University Lisbon, Portugal, in 2007 and 2012, respectively. He is an Associate Professor (Biträdande Professor) in manufacturing engineering, with a specialization in industrial cyber-physical systems, with Linköping University, Sweden. His Habilitation (SE Docent) is in manufacturing engineering from Linköping University. He has participated in several national and international research projects in the field of intelligent and plug and produce production systems, and was a Freelance Consultant and a Software Engineer in the automotive industry. He has (co)authored more than 80 papers and chapters published in journals, books, and conference proceedings and is the author of the book *System Design and Implementation Principles for Industry 4.0*. His research interests include the modeling, development and implementation of intelligent software solutions-based in multi-agent systems, service-oriented architectures, and cloud for the fast reconfiguration, execution, and ramp-up of batch-size-one production systems. He is a member of the Swedish Advisory Production Council. He is also the Vice-Chair of the IEEE Technical Committee on Industrial Agents.

LUIS GOMES (Senior Member, IEEE) received the Electrotech. Engineering degree from the Technical University of Lisbon, Portugal, in 1981, and the Ph.D. degree in digital systems from NOVA University Lisbon, Portugal, in 1997. From 1984 to 1987, he was with EID, a Portuguese medium enterprise, in the area of electronic system design, at the Research and Development Engineering Department. He was made the Honorary Professor of the Transilvania University of Brasov, Braşov, Romania, in 2007, as well as the Honorary Professor of Óbuda University, Budapest, Hungary, in 2014. He is an Associate Professor, with habilitation, at the Electrical and Computer Engineering Department, NOVA School of Science and Technology, NOVA University Lisbon, and a Researcher at the Center of Technology and Systems, UNINOVA Institute, Portugal. He is an author/coauthor of more than 300 papers and chapters published in journals, books, and conference proceedings, as well as a (co)author of one book and a co-editor of three books. His main research interests include the usage of Petri nets and other models of concurrency, applied to reconfigurable and embedded systems co-design and cyber-physical systems. He received the IEEE Industrial Electronics Society Anthony J. Hornfeck Service Award, in 2016.