International Journal on Interactive Design and Manufacturing (IJIDeM) (2023) 17:525–538
https://doi.org/10.1007/s12008-022-00968-0

ORIGINAL PAPER

Revisiting requirement engineering for intelligent manufacturing

Javier Martinez Silva[1] · Pedro Manuel Gonzalez del Foyo[2] · Arianna Zoila Olivera[3] · Jose Reinaldo Silva[4]

Received: 31 January 2022 / Accepted: 5 May 2022 / Published online: 30 June 2022

Abstract
The upcoming challenge for innovation in intelligent manufacturing design introduced a demand for agility and flexibility, adapting production plants and processes to emerging needs and new services. Arrangements of service-oriented cyber-physical components should replace computer-integrated plants, matching anthropocentric production lines. Manufacturing should follow a process-oriented requirements cycle, linking different design phases based on traceability, associating problem, solution, and collaboration with external and human agents. The first problem is how to provide a requirement cycle that fits this demand, using a systemic and process-oriented (formal) method. This article proposes a model-based requirements cycle for intelligent manufacturing systems (IMfgS). The proposal covers a functional, object-oriented approach and introduces a goal-oriented method suitable for service design. Process orientation leads to Petri Nets' schema, already used in plant design manufacturing. The Petri Net formal approach synthesizes requirements and describes solutions, opening a possibility to trace problems and solutions. A case study from the chemical industry illustrates that. A requirements life-cycle formalized in Petri Nets includes transference algorithms from either UML or KAOS diagrams. The approach can be adapted to service-oriented manufacturing, but that is not developed formally in this work. The requirements cycle has been adjusted to available tools, making the proposal practical. Intelligent manufacturing is getting more attention, either because of demands for sustainable manufacturing processes or the digitalization process and industry 4.0. New design processes demand more flexibility and capacity to reuse and modify functions while also modifying the product/services they produce. New approaches recover methods typically used in Software Engineering. However, such processes also bring complexity and the need for intensive, interactive testing, even during the requirements phase.

Keywords Requirements engineering · Intelligent manufacturing · Requirements development · Manufacturing design · Interactive design

These authors contributed equally to this work.

✉ Jose Reinaldo Silva
reinaldo@usp.br

Javier Martinez Silva
javier.silva@sidia.com

Pedro Manuel Gonzalez del Foyo
pedro.foyor@ufpe.br

Arianna Zoila Olivera
azoliverar@gmail.com

1 SIDIA, Institute of Science and Technology, Manaus, Amazonas, Brazil

2 Mechanical Eng., Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil

3 TrueChange, Co., Recife, Pernambuco, Brazil

4 Escola Politecnica, Universidade de São Paulo, São Paulo, Brazil

1 Introduction

Manufacturing (Mfg) is going through a disruptive change where mass-production automated systems are gradually replaced by models no longer coupled with itemized products. Integrated manufacturing evolves by adding new sensors, computing platforms, communication, and power supervisory systems. New concepts such as Cyber-Physical Systems (CPSs), twin systems, intensive modeling, Industrial Internet of Things (IIoT), and big data [1,2] complete the transformation. Many of those changes affect directly the relationship between workers and the automated production lines [3], launching new design approaches [4–8]. The real challenge is anticipating requirements modeling's formalization - especially if the target is an automated and service-oriented manufacturing (Mfg) process. In such cases, it is crucial to include a model for the coupling with customers in those formal requirements.

Process-oriented requirements engineering is the key to achieving a formal approach that includes customer coupling. Generically, the Requirements Engineering (RE) phase consists of eliciting, modeling, and verification steps. A formal presentation is not (conventionally) introduced in the modeling steps. In its turn, Intelligent Manufacturing requirements are dynamic; that is, they would actively influence the design, and after that, the implementation and deployment[9,10] by providing support to maintenance. In our proposal, dynamical requirements are represented by invariants or background rationales.

This paper proposes a life cycle for requirements engineering applied to automated, intelligent manufacturing arrangements with the following characteristics:

- Requirements are dynamic;
- Requirements are formally modeled using language representations suitable to express the dynamic of automated problems, such as Petri Nets;
- Requirements are formally analyzed and verified in each step of the cycle;
- Requirements should include customer coupling;
- Requirements should be the basis for innovation and maintenance, referring to requirements to include new features.

Petri Nets (PNs) emerged as a suitable and formal presentation to dynamic requirements [11]. Different Petri Nets proposals appeared since its proposition in 1962 until the appearance of the ISO/IEC 15.909 (www.iso.org /standard/67235.html) standard. The standard defines the basic types: Place / Transition, High Level, and Asymmetric nets (15.909-1). An interchange format PNML (Petri Net Markup Language) (15.909-2) may allow a formal model to be shared by different tools. User extensions (15.909-3) are accepted since they respect the formal definitions of 15.909-1. Extensions could be hierarchical nets, fusion places and transitions, gates, time, and object-oriented nets.

Therefore, it is encouraging to use Petri Nets as a formal presentation to requirements, fitting dynamic analysis and intelligent issues derived from draft models in diagrammatic (semi-formal) languages. UML is the most popular diagrammatic language (used in different project domains). Consequently, it is a suitable candidate to support new approaches for requirements analysis. Recent proposals have emerged, offering alternatives to functional development. The goal-oriented method is the most promising in this line[12].

Goal-oriented requirements engineering (GORE) emerged from the search for alternatives to strictly functional approaches [13]. It evolved to schematic presentations like KAOS [12]. Goal-oriented diagrams can be transferred to Petri Nets by specific algorithms. The authors of this paper developed a software tool called ReKPlan (Requirement Engineering in KAOS for Planning) to transfer KAOS diagrams automatically to Petri Nets [14].

The following section will show some basic concepts of Petri Nets and how we intend to use them to formalize the requirements of intelligent manufacturing systems modeled in UML. A small example of evaporator control will illustrate the process. Section 3 will discuss the proposed requirements life cycle using UML, followed by an alternative goal-oriented approach. A case study from Roadef Challenge, a Car Sequencing problem [14], will be reproduced to highlight these two different approaches' differences. Section 5 will analyze the current context for manufacturing design and perspectives to match the proposed requirements life cycle with tools commercially available. Finally, Sect. 6 will show concluding remarks and perspectives on further work.

2 From draft to goal-oriented requirements

The requirements process's efficiency could make the difference between success and failure in systems design [15], especially for automated system domains. The requirements cycle should deliver a consistent set of statements defining a closed vision of the problem, even if the description is not complete - we would call that a model. It should also be dynamic and guide the project's evolution to implementation, including maintenance and final destruction. The term "dynamic requirements" has been used since the 50s, applied to different disciplines and domains with this same meaning[16]. More recently, with the prevalence of functional design, its use was reduced, particularly in automated systems. However, the emergence of service-oriented manufacturing and the demand for traceability brought back the need to reinforce dynamic requirements again [16]. The design of service systems (or product-service systems) demands recurrent checks for the satisfiability of requirements in different life cycle phases.

The early requirements phase must overcome two basic problems:

- The *genesys problem*, which states that requirements are derived from intentions and distinct (and sometimes conflicting) viewpoints, which cannot be completely formalized;
- The *scope creep* when emerging requirements affect the problem domain being used.

A requirement cycle that transfers diagrammatic and semi-formal representation into a formal language would face both problems. In this work, dynamic requirements should be represented by *invariants*, formalized using Petri Nets [17].

To face the *genesys problem*, we assume that the elicitation process starts by capturing intentions and demands (from different viewpoints) in clustered semi-formal diagrams. UML is a prominent example used in many object-oriented structured elicitation processes. It is used in this paper as a reference. That means to make the life cycle iterative proposal strictly a functional approach first, and then propose the evolution to other methods (as the goal-oriented).

The requirements life cycle proposed in this paper starts with a preliminary UML model initiating a cycle where each model is analyzed and formalized. Models express system dynamics by a state-transition (or Transition Systems) schema both in requirements and the solution. That means capturing semi-formal diagrams into Petri Nets [14,17].

2.1 The Petri nets formalism

Petri Net was created in 1962 with Carl Adam Petri's doctoral thesis, which focused on process management and communication between processes. With a hybrid background in Mathematics and Computer Science, Petri based his formalism on bipartite graphs and transitions systems - a key concept to computation theory. Such combination results in a sound formalism that is also executable, an important criterion to guide a formal presentation applied to Engineering Design. Formally[18],

Definition 1 (*Petri Net*) A Petri net structure is a directed weighted bipartite graph

$N = (P, T, A, w)$

where

P is a finite set of places, $P \neq \emptyset$
T is a finite set of transitions, $T \neq \emptyset$
$A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs from places to transitions or from transitions to places
$w : A \rightarrow \{1, 2, 3, \ldots\}$ is a mapping between the set of arcs and the positive integers.

We will normally represent a set of places by $P = \{p_1, p_2, \ldots, p_n\}$ and a set of transitions by $T = \{t_1, t_2, \ldots, t_m\}$, where $\mathtt{P} = n$ and $\mathtt{T} = m$ are the cardinality of the respective sets. A typical arc is an ordered pair (p_i, t_j) or (t_j, p_i), deppending on its direction. A mapping can associate a non-zero positive integer to its arc denoting a weight w, that is, the amount of tokens required to flow in the arc.

The authors developed a Petri Net modeling environment called GHENeSys (General Hierarchical Enhanced Net System) to model Place/Transition and High-Level Nets[1] following the standard ISO/IEC 15.909. A user extension introduces fusion places, inhibitor gates, hierarchy, and timed models. The hierarchical extension follows the Theory of Structured Programming [19]. Hierarchical elements are called macro-box and macro-activities and can be replaced by homogeneous sub-nets. Borders elements belong to the same type, either place or transition. We have then *proper* elements. These elements are said *live* if there is at least one path between two input and output border elements without deadlock. In other words, there is at least one free path leading from input to output. In this case, hierarchical levels preserve properties [18].

Definition 2 (*Macro-place* (*transition*)) A macro-place (transition) is a substitution place (transition) element [20] that can replace a place-bounded (transition-bounded) sub-net which is a proper entity [19], that is, there is only one input e one output element and at least one path between then without traps or deadlocks.

Definition 3 (*Pseudo-boxes*) A special place element, a pseudo-box, is a fusion place to match transfer signals provided by other sub-nets - reinforcing the structured approach - or external signals from the surrounding environment. Those signals are observed but not controlled by the system being modeled.

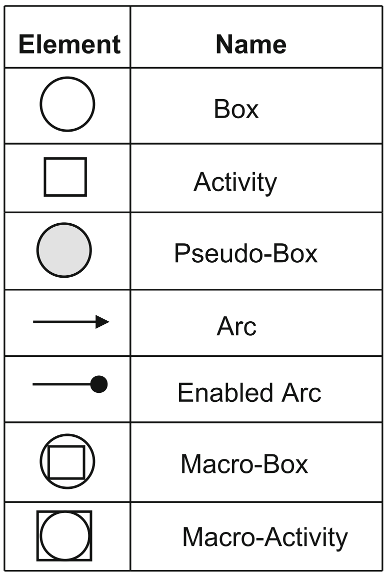
GHNeSys can be used for basic modeling and support formal verification (model checking) using Petri Nets and GHENeSys extension. Some of its essential elements are depicted in Fig. 1.

In Fig. 1, Box and Activity are perfect matches to classic place and transitions, respectively, but accept hierarchical static (Macro-Box) and dynamic (Macro-Activity) definitions. The Pseudo-Box is a fusion place that denotes the current sub-system's connection with the application domain elements or other sub-systems. Arcs are a pictorial representation of the flux relation. Enabling Arcs are generally associated with Pseudo-Boxes and inserted without destroying the state equation. They are also helpful in expressing external connections.

2.2 Mapping UML to Petri nets

The number of articles dedicated to the translation of UML diagrams to Petri Nets increased significantly since the beginning of this century [21–26] due to the growing interest in formalizing the design process. Although different sets of

[1] While a classic net set of tokens standing for items of control flow, a High-Level Net admit static properties that can distinguish each mark.



Element	Name
○	Box
□	Activity
●	Pseudo-Box
→	Arc
—●	Enabled Arc
⧈	Macro-Box
⧇	Macro-Activity

Fig. 1 GHENeSys is a unified extensions to the basic Place/Transition and High-Level definitions, including elements as Pseudo-box - representing observable but not controllable events -, hierarchy and time

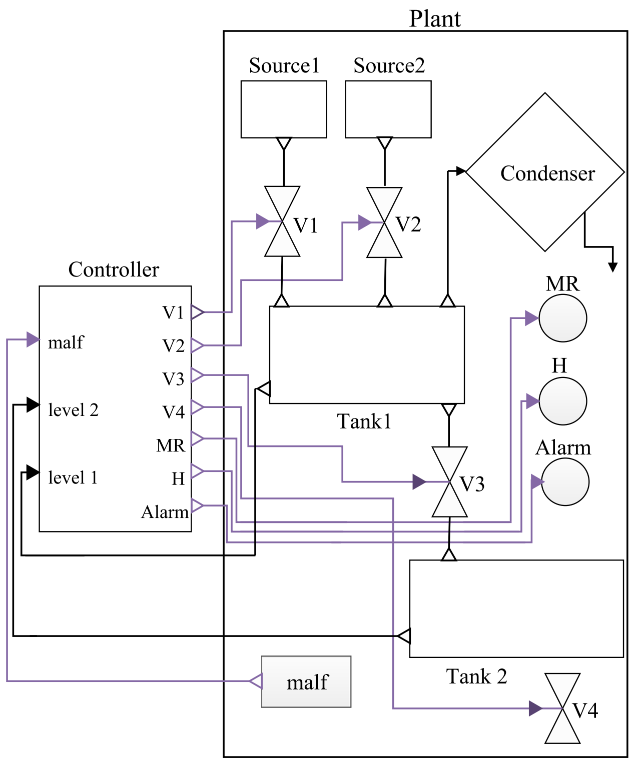


Fig. 2 Schema for the evaporator

UML diagrams have been used in the transfer, all share the idea of selecting a minimal set.

We improved Baresi classic algorithm for translation from UML to Petri Nets [21] in [17]. An illustrative (and realistic) example will show this translation using an extended set of diagrams: class, state, and activity translated to a Place/Transition net using GHENeSys. The example is based on an evaporator system controller's specification, shown in Fig. 2.

The evaporator system works according to the following (functional) description:

- Tank 1 starts to be filled by opening valves V1 and V2;
- Valves V1 e V2 are closed when Tank 1 is full;
- The Mixer turned on;
- After two time unit intervals, the Heater is turned on and actuates for 20 time units. The solution evaporates and is cooled in the Condenser.
- The Heater is turned off;
- The solution in Tank 1 is transferred to Tank 2 by opening valve V3;
- The Mixer is turned off if Tank 1 is empty and valve V3 is closed;
- The solution must remain in Tank 2 for 32 time units and then be liberated by opening valve V4.
- Valve V4 is closed when tank T2 is empty, and the system returns to the initial state.

As established before, the functional approach is accurate but does not cover non-functional requirements. We might want to insert exception or failure possibilities modeled in a semi-formal language like UML.

UML diagrams represent the controller's intended behavior (requirements), including exceptions. State and class diagrams are shown in Figs. 3 and 4 respectively [2].

Briefly, let us suppose that the required initial conditions are not valid. Tank 1 is expected to be empty, V3 is closed, the system has no alarm set, and there is no indication of a malfunction in the mixer or the Heater. In that case, the sequence described above goes smoothly. Otherwise, the malfunction state is set, the alarm is turned on (if it is not on already), and when tanks T1 and T2 are empty, it is turned off and returns to the initial state.

Although the functional, intuitive description is purely sequential, there are parallel actions, as shown in the Petri Net representation.

Salmon et al. [17] proposed a UML-PetriNet translation algorithm, based on Baresi's work [21] [3]. The net resulting from this process is in Fig. 5. The diagrams of Figs. 3 and 4 were the input for this process.

We are not going into details about the translation algorithm, but it is available in a previous work [17]. Requirements development based on UML/PN transference depends

[2] Since it is only an illustration, we are focusing only on the controller.

[3] Baresi proposed a direct matching, a bijection between UML elements and Petri Nets localities.

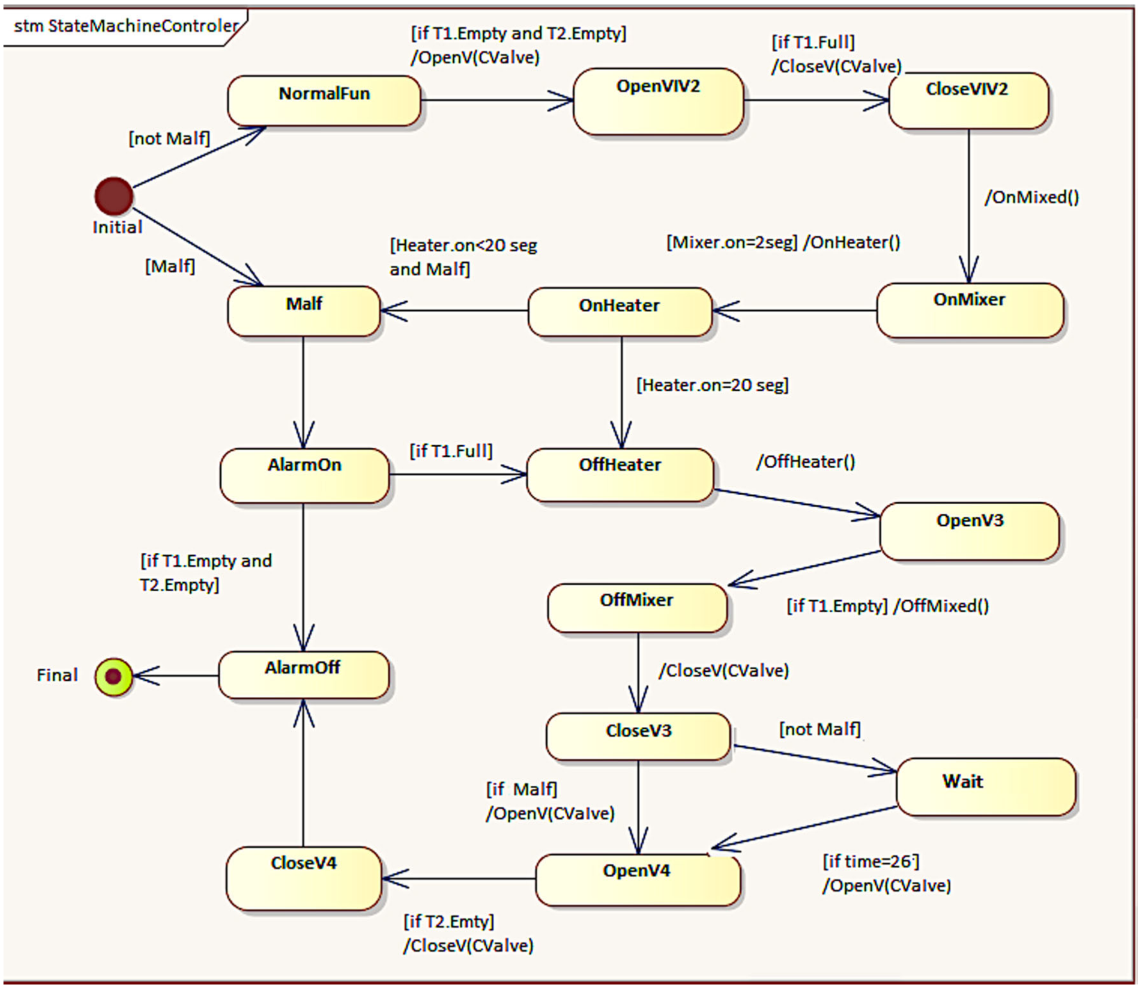

stm StateMachineControler
Initial
[not Malf]
[Malf]
NormalFun
[if T1.Empty and T2.Empty] /OpenV(CValve)
OpenVIV2
[if T1.Full] /CloseV(CValve)
CloseVIV2
/OnMixed()
OnMixer
[Mixer.on=2seg] /OnHeater()
OnHeater
[Heater.on<20 seg and Malf]
Malf
[Heater.on=20 seg]
AlarmOn
[if T1.Full]
OffHeater
/OffHeater()
OpenV3
[if T1.Empty] /OffMixed()
OffMixer
/CloseV(CValve)
CloseV3
[not Malf]
Wait
[if time=26] /OpenV(CValve)
[if Malf] /OpenV(CValve)
OpenV4
[if T2.Emty] /CloseV(CValve)
CloseV4
[if T1.Empty and T2.Empty]
AlarmOff
Final

Fig. 3 State Diagram for the Controller

on the designer's ability to capture non-functional requirements. However, it can synthesize dynamic requirements, that is, requirements that could be used as a reference during the whole design process up to deployment.

Table 1 shows some dynamic requirements, which justifies the use of Petri Nets formalization for the requirements modeling, providing an executable language presentation used in formal verification. OCL (Object Constraint Language) is also used to describe dynamic requirements [27]. In the proposed functional requirements development, OCL sentences complement functional requirements, adding constraints from the application domain, non-functional requirements, or invariants. The following section will present the life cycle based on a functional, object-oriented approach.

3 The requirements life cycle

The Petri Net in Fig. 5 was obtained by an algorithm [17], using a minimal set of UML diagrams. The transference composes the proposed functional requirement life cycle, depicted in Fig. 6.

Table 1 Dynamic requirements for evaporation matched with Petri Nets invariants

Id	OCL Specifications
1	$(Tank1.Empty, Tank2.Empty) \rightarrow V1V2.open$
2	$Tank1.Full \rightarrow Mixer.On$
3	$Tank1.full \rightarrow V1V2.close$
4	$Tank1.Empty \rightarrow V3.close, Mixer.off$
5	$Tank2.full \rightarrow V4.open$
6	$Tank1.Empty, Tank2.Empty, malf.false) \rightarrow alarm.off$

The process starts with eliciting a set of requirements (from distinct viewpoints), checking for the generation gap[4], and generating a preliminary model. This semi-formal model (Fig. 6) is formalized by transferring it to Petri Nets. Prop-

[4] Missing requirements as occurs typically in early elicitation

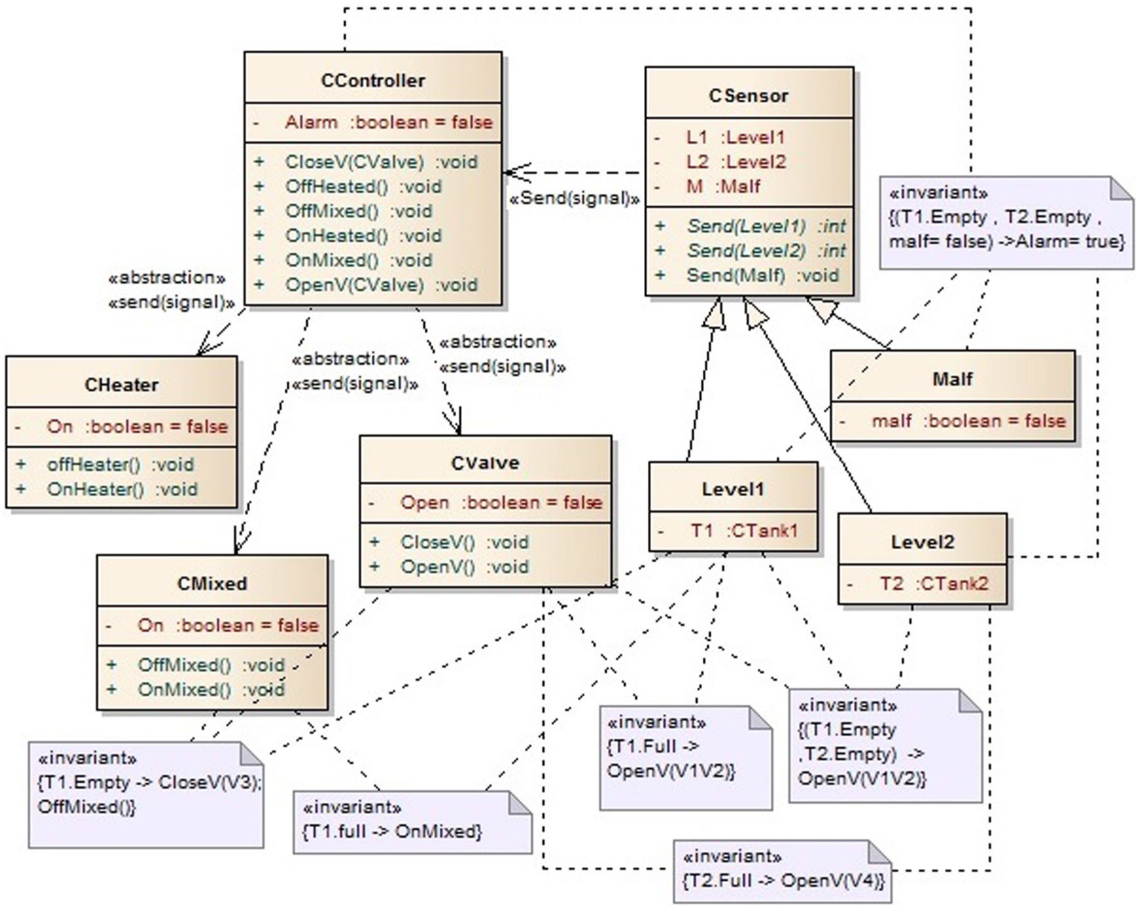


Fig. 4 UML Class Diagram for the Controller

erty analysis over the formal model will check for invariant preservation, consistency, and closeness. Eventually, the requirements could be verified and matched with general (dynamic) requirements proposed in the definition of the project. A revision process will look for gaps that justify enhancing the requirement's model, initiating another cycle. When no enhancement is feasible, the requirement specification is ready for documentation and to go forward in the design process.

The requirements cycle will be essential even in traditional product-oriented manufacturing automation. Nevertheless, it becomes more critical to design product-service systems (PSS)[28,29]. Other formalization methods can be used to generalize the application domain, such as Event-B [30]. However, Petri Nets would better represent generic systems and process-oriented Model-Based Requirements Engineering (MBRE). Petri Nets can provide a formal state/transition approach where concurrency is analyzed and applied to distributed systems.

Appying the cycle (Fig. 6) to the evaporator control requirements (described by UML diagramas in Figs. 3 and 4), we transferred requirements to the Petri Nets shown in Fig. 5.

Of course, the practical use of the methods discussed here would require the requirement cycle's encapsulation in a software tool that helps with process scaling and documentation. Features such as re-usability and version control are essential to achieve better results in practice. This paper intends to focus only on the method and its theoretical background. Still, we will discuss some ways to make it practical in the last section.

The requirements cycle in Fig. 6 relies on a cyclic refinement formal process that applies to functional and goal-oriented approaches. However, scaling that to large distributed production systems would bring some problems (supposing UML as the requirements presentation):

- A demand to reduce redundancy and use a minimal set of diagrams;
- Dealing with large packages of subsystems;
- Including restrictions and conflicts between elements and subsystems;

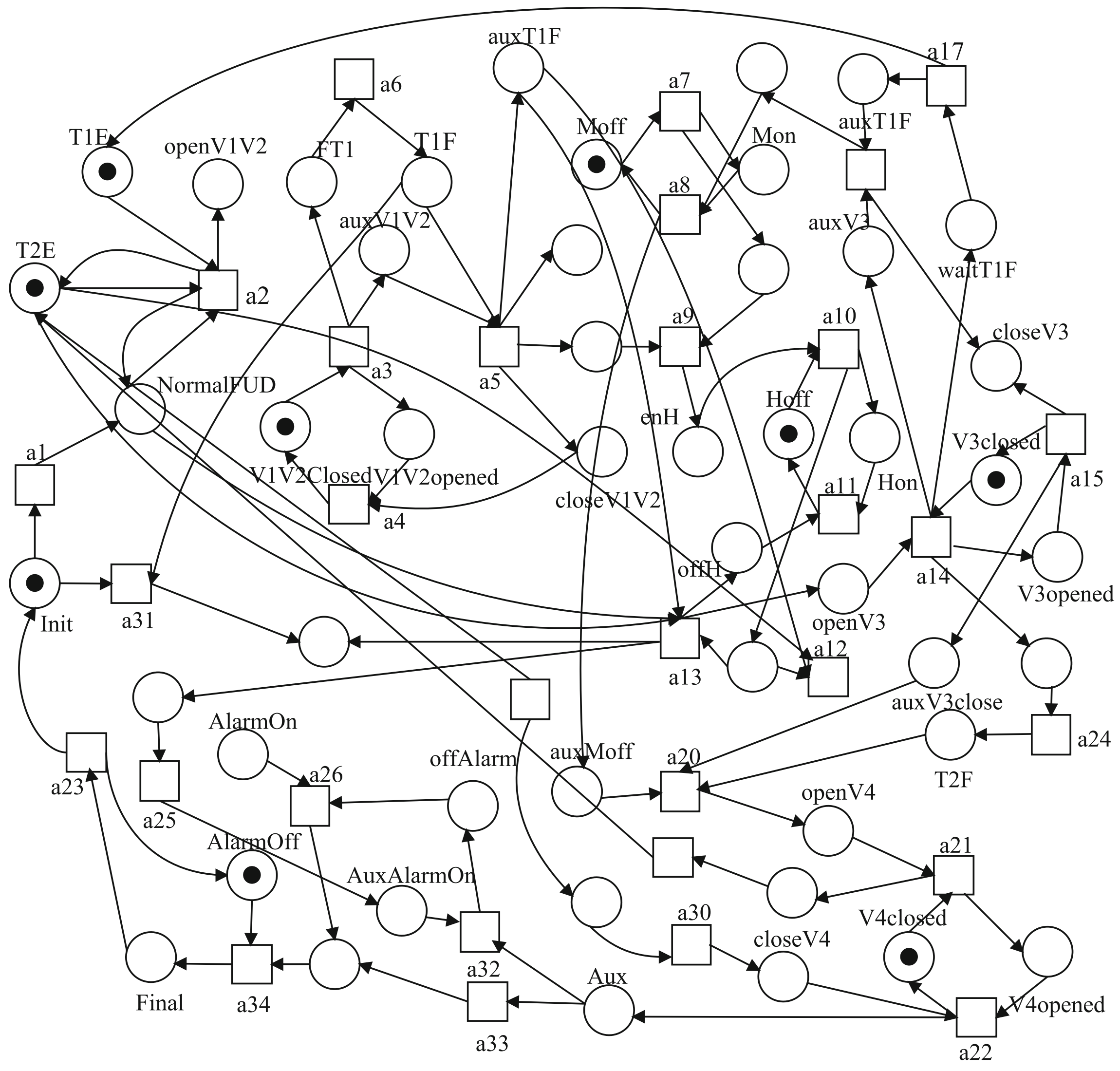


Fig. 5 Petri Net synthesized from UML diagrams for the Controller

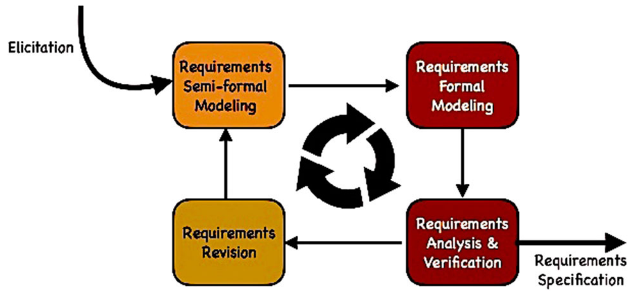

Elicitation
Requirements Semi-formal Modeling
Requirements Formal Modeling
Requirements Revision
Requirements Analysis & Verification
Requirements Specification

Fig. 6 The proposed requirements development cycle

- Transferring hierarchical semi-formal presentation in UML to hierarchical Petri Nets.

Therefore, there are enough reasons to look for alternative approaches to formalize the requirements process. A different approach is obtained using KAOS (a goal-oriented approach) as semi-formal presentation - which also is suitable to represent service-oriented manufacturing [4,31].

4 Goal-oriented requirements engineering (GORE)

The increasing importance of requirements in systems design [15] encourages the search for alternatives to replace a strictly functional approach, both on software [32] and manufacturing [33] design. An alternative approach emerged based on goals, instead of functionalities [34][5].

[5] The focus on goals can be retraced to C. Rolland and C. Souveyet paper: Structured Analysis of Requirements Definition, IEEE Trans. on Software Engineering, vol.3, num. 1, 1977.

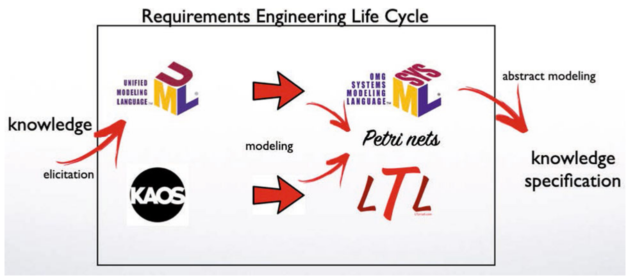


Fig. 7 The general requirements life cycle and the introduction of Petri Nets to formal analysis

Goal-Oriented Requirements Engineering (GORE) was formalized by John Myloupolos [35] and Axel Lamsweerde [12]. It inspired several works in requirements engineering. Focusing on goals instead of objects or functionalities, GORE postpones the concern with non-functional requirements, making requirements modeling more flexible and secure. However, GORE's distinct advantage is the suitability to treat the coupling between system and users, making it more adherent to service-oriented design. This characteristic also points to GORE as a suitable approach to designing intelligent manufacturing systems [4,6].

Visual modeling is covered in GORE by a semi-formal representation called KAOS. The following subsection will give a brief description of KAOS diagrams.

4.1 KAOS diagrammatic representation

KAOS representation comprises four related (but not redundant) modeling diagrams: goal diagram, object diagram, operation diagram, and responsibility diagram. A goal diagram is a tree in which all nodes except the root represent related subgoals, and edges represent relations (such as composition, refinement, dependency, or restriction). Project statements are related to the main goal (the root). Fig. 7 shows the core elements of KAOS diagrams. (Fig. 8)

Goals should be achieved by *requirements*, which, in its turn, are associated with operations. Requirements should fit informal demands or expectations from specific users or stakeholders. Thus, it is possible to implicitly represent the merging of different viewpoints addressed by various agents' classes attributing their respective responsibilities. That is the basis for traceability, which relies on invariants and dynamic requirements concepts.

Requirements modeling using KAOS or UML differs significantly from the approach directed to functionalities :

- In the number of diagrams: four to KAOS, instead of thirteen structural and twelve behavior diagrams in UML. The diversity of diagrams encourages searching for methods to reduce it to a minimal set;

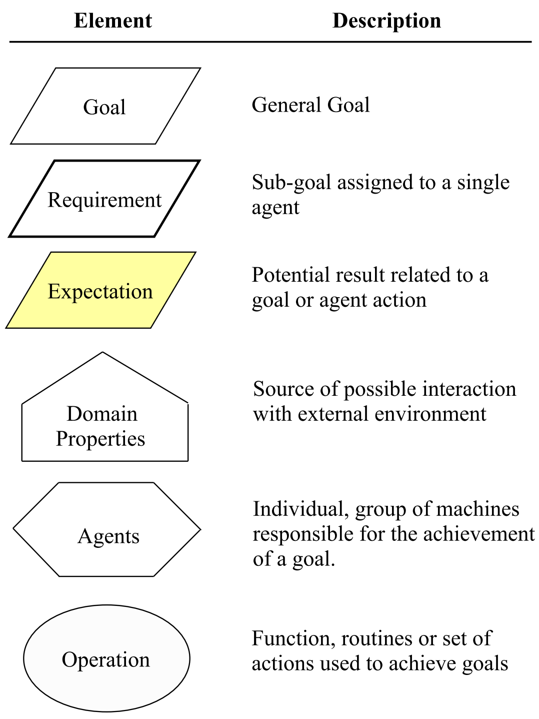

Element
Description
Goal
General Goal
Requirement
Sub-goal assigned to a single agent
Expectation
Potential result related to a goal or agent action
Domain Properties
Source of possible interaction with external environment
Agents
Individual, group of machines responsible for the achievement of a goal.
Operation
Function, routines or set of actions used to achieve goals

Fig. 8 Main elements of Goal Diagram

- Using time in requirements analysis to schedule activities turn the functional approach a challenge [36];
- Representing workflow is also an issue for manufacturing design, not fully assisted in UML or KAOS.

Linear Temporal Logic (LTL) is customarily used to formalize KAOS diagrams. However, the best way to put together logic-proof obligations, workflow, and eventually time constraints (or even real-time) is to use a widely formal graph representation, such as Petri Nets, capable of expressing concurrency and workflow and time extensions.

The following section will show an alternative requirements cycle replacing the semi-formal UML representation with KAOS. Algorithms already developed for some of the authors to transfer UML to Petri Nets [17] will be replaced by a new one (also developed by the authors) to transfer KAOS diagrams to Petri Nets.

4.2 Using Petri nets to analyze requirements

The proposed requirement cycle is process-oriented and selected Petri Nets as a formal approach since it is widely pointed to as suitable for this kind of application [37–39].

Petri Net is executable, which suits even better the requirement engineering process. A specific implementation of Petri Nets that follows the standard ISO/IEC 15.909 is used to formalize and verify requirements. The modeling platform is called GHENeSys (General Hierarchical Enhanced Net System) [18].

GHENeSys can work with classic P/T Nets or High-Level Nets as a unified net system. Extensions can also include hierarchy, time slice, time, and specific fusion places.

We propose an algorithm to transfer goal-oriented diagrams to classic or High-Level Petri Nets. Analysis and verification can then be performed, including invariant analysis. Invariants are defined during the initial requirements life cycle (RLC) or after semi-formal analysis and verification. Generally, emerging requirements are discovered during the net analysis and included in the requirements cycle.

Transfering KAOS to Petri Net implies a direct matching between diagrams and net regions. We propose a translation between a KAOS Markup Language (KML) and PNML - the XML description of Petri Nets. An algorithm based on heuristics [14] performs the transference from KML to PNML (Petri Net Markup Language)[40]. We will use a markup language associated with a unified extended Net GHENeSys (General Hierarchical Enhanced Net System) depicted in Fig. 9.

An illustration of a step in the requirements cycle using a goal-oriented approach is given in the next section using a problem proposed by the automotive industry.

4.3 Case study: A car sequencing problem

General requirements for this problem are: the Car Assembling process must be flexible enough to fulfill particular and personalized demands elicited from final users [41] or to introduce requested items such as sunroof, customized wheels, air conditioning, or other similar items. Customer orders generally arrive at car factories in real-time and must be assigned to production with severe deadline constraints. A car sequence must be established daily, and its accuracy directly affects the amount and quality of cars delivered. (Nguyen, 2005) [42] shows a detailed description of demands for the production process.

The challenge is to propose a digital system that collaboratively controls the sequence lines using the requirements life cycle described in Fig. 6. We will use an alternative approach based on KAOS to capture requirements and transfer them to Petri Nets for formal analysis and verification. The transference between these representations is based on the match depicted in Fig. 9.

We developed a software tool called ReKPlan (Requirement Knowledge for Planning) to capture goal and object diagrams. Figure 10 has a snapshot of ReKPlan tool [14].

Synthesis of Petri Net comes from KAOS, or LTL equations[6]. Table 2 shows the LTL expressions for the car

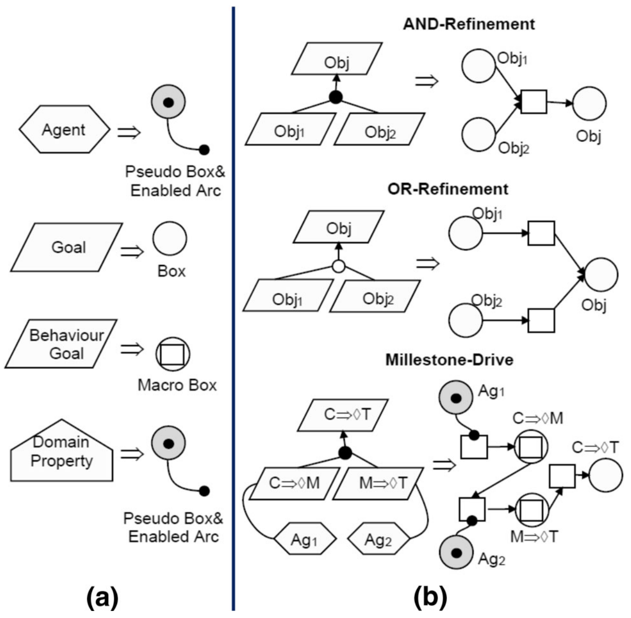


Fig. 9 **a** Translation for basic elements. **b** Translation for refinements

sequencing problem. The Petri Net representation verifies properties such as closeness, consistency, deadlock, fairness, and invariants using the GHENeSys environment.

5 Perspectives for practical use of the requirements cycle

The requirements cycle of Fig. 6 (either using UML or KAOS development line) must be encapsulated in a software environment to become useful in real projects. We implemented RekPlan only to test basic features and reinforce the applicability of the requirement cycle. Additional features and integrated user experience are necessary to turn it into an environment and practical method. Some of these issues are:

- Add database support to store semiformal requirements, linked to the respective formal representation: SysML, LTL, or Petri Nets;
- Add knowledge-based support for reusability of modules or sub-systems;
- Extends the formal approach to support real-time requirements and modeling[7];
- Introduces version control and non-relational data structuring to extract better performance for traceability and check dynamic requirements (invariants);
- Improves algorithms to automate the transference from semiformal representation (UML or KAOS) to Petri Nets;

[6] Software tools commercially available can automatically derive LTL expressions. Objectiver (www.objectiver.com) is one of them

[7] UML uses timeline diagrams, but KAOS does not have any formalization for time dependency.

Table 2 LTL sentences associated to some goals of Goal Diagram

Goal	LTL Sentences
Cars painted when an order was recei - ved.	$\forall c$:Car, $\exists pa$:PaintingArea,$painter$: Painter, sg : SprayGun,$color$:Color; **isOnPA**$(c,pa) \wedge$ **sprayGunInPA**$(sg,pa) \wedge$ **use**$(sg,color) \wedge$ **paint-Color**$(c,color) \wedge$ **workingInPA**$(painter,pa) \wedge \neg$ **painted**$(c) \wedge c$.**posPainting** = $painter$.**last-Painted** $+1 \Rightarrow \diamond$ $painter$. **last-Painted**= c.**pos-Painting** $\wedge$ gs. **sprayGunLimit**=gs. **sprayGunLimit**$+1 \wedge$ **painted**(c)
Cars assembled while painted in painting area.	$\forall c$:Car,$\exists ass$:Assembler, aa :AssemblingArea; **painted**$(c) \wedge$ **isOnAA**$(c,aa) \wedge$ **groupedAssem-bled**$(c) \wedge$ **workingAA**$(ass, aa \wedge \neg$ **assembled**(c) $\wedge$ c.**posAssembling**= ass.**lastAssembled**$+1 \Rightarrow \diamond$ mnt.**lastAssembled**= c.**posAssembling** $\wedge$ **assem-bled**(c)
Cars grouped according spray gun limit.	$\forall$ c:Car, $\exists$ op:$Operator$; $\neg$ **painted**$(c) \wedge \neg$ **assem-bled**$(c) \wedge$ **availableOperator**$(op) \wedge c$.**posPainting** $= 0 \Rightarrow \diamond$ **groupedPaint**(c)
Cars grouped by special features	$\forall c$:Car, $\exists$ op:$Operator$; **painted**$(c) \wedge$ **available-Operator**$(op) \wedge \neg$ **groupedAssembled**$(c) \Rightarrow \diamond$ **groupedAssembled**(c)
Cars transported to painting when grouped.	$\forall c$:Car, $\exists tra$:Transporter, $\exists pa$:PaintingArea, sg :SprayGun; **groupedPaint**$(c) \wedge$ **available-Transporter** $(tra) \wedge \neg$ **painted**$(c) \wedge$ $\neg$ **isOn-PA**$(c,pa) \wedge$ pa.**currentPaint** $<$ sg.**sprayGunLimit** $\Rightarrow \diamond$ **isOnPA**$(c,ap) \wedge$ pa.**currentPaint** $=$ pa.**currentPaint** $+1 \wedge$ c.**posPainting** $=$ pa.**currentPaint**
Cars transported to assembling when grouped.	$\forall$ c:Car, $\exists tra$:Transporter, aa :AssemblingArea; $\neg$ **assembled**$(c) \wedge \neg$ **isOnAA**$(c,aa) \wedge$ **avail-ableTransporter**$(tra) \wedge$ **groupedAssembled**$(c) \Rightarrow$ $\diamond$ **isOnAA**$(c,aa) \wedge$ aa.**current- Assembled** $=$ aa.**currentAssembled** $+1 \wedge$ c.**posAssembling** $=$ aa.**currentAssembled**
Spray gun washed with solvent.	$\forall$ sg :SprayGun, $\exists$ painter:$Painter$; $\neg$ **clean**$(sg) \wedge$ **has**$(painter, sg) \wedge painter$.**qcarsPainted**$> 0 \Rightarrow \diamond$ **clean**$(sg) \wedge painter$.**qcarsPainted**$= 0$
Spray gun ready to paint after it was washed.	$\forall$ c:Car, sg:$SprayGun$, $\exists$ painter:$Painter$; **has**$(painter, sg) \wedge$ **clean**$(sg) \Rightarrow \diamond \neg$ **clean**(sg)

- Builds a design environment in the cloud, reinforcing collaborative work supported by data analysis.

Besides the practical aspects, a helpful system design tool for intelligent manufacturing should also cover the tendency to address Product-Service Systems (PSS), an ultimate goal for Industry 4.0.

5.1 Intelligent manufacturing as product-service systems

The idea of a symbiosis between product and service in a unique artifact appeared at the beginning of this century in works that pointed towards a way to insert sustainability in projects, especially in production lines [43]. Initially, the target was to introduce “service features,” called servitization, associated with vertical integration, combining manufacturing and business processes. This approach evolved into a product-service notion related to information systems and manufacturing resources planning (MRP) Architecture[44].

Service was a perfect match for information systems, and a Service-oriented Modeling Framework (SOMF) was proposed by Michael Bell [45]. The framework was fully implemented in the Enterprise Architect (www.sparx systems.com) platform and used to model requirements for new information systems. SOMF was extended later to deal with a broader concept of service (which would include manufacturing services) and implemented in the same Enterprise Architect platform (but not commercialized) [46].

The next challenge is to achieve a complete implementation of the product-service design, including a definition of the coupling with different classes of users (in a pro-

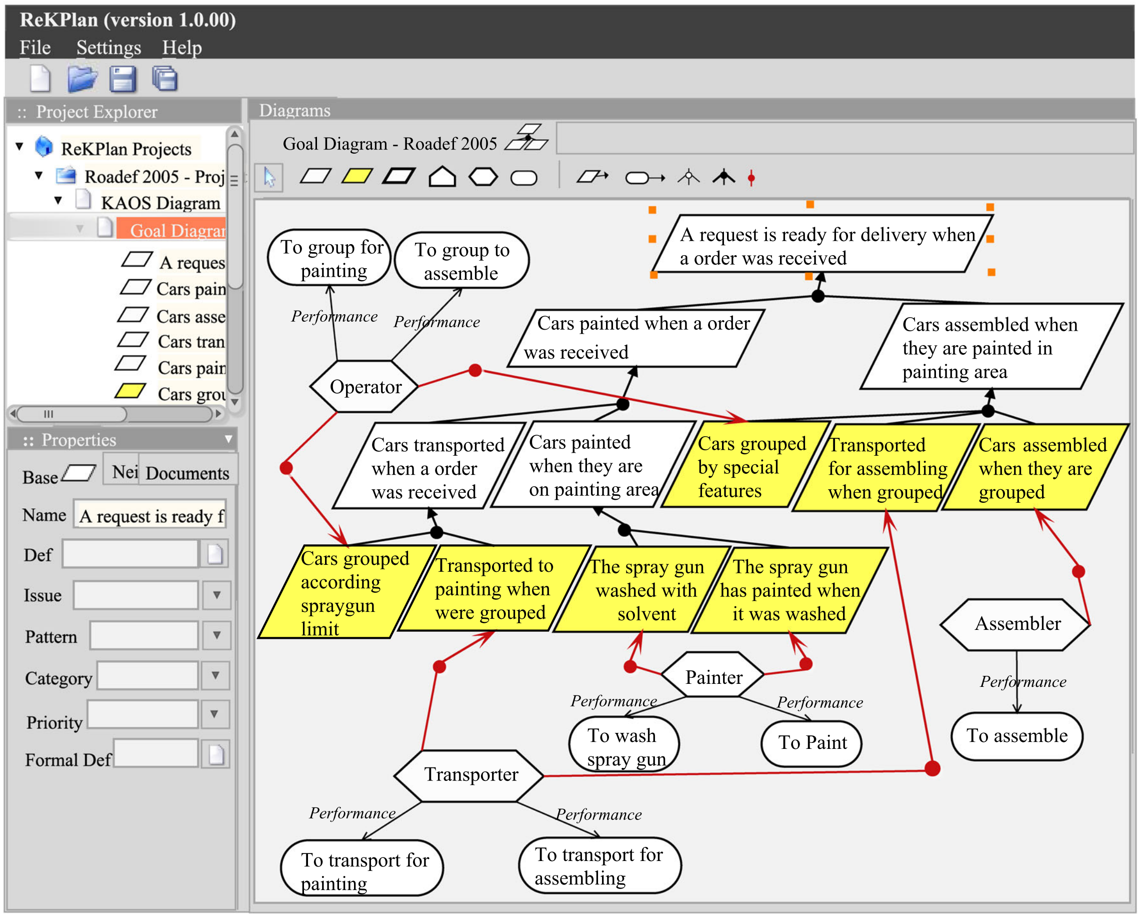

ReKPlan (version 1.0.00)
File Settings Help
:: Project Explorer
ReKPlan Projects
Roadef 2005 - Proj
KAOS Diagram
Goal Diagrar
A reques
Cars pain
Cars asse
Cars tran
Cars pair
Cars grou
:: Properties
Base
Nei Documents
Name A request is ready f
Def
Issue
Pattern
Category
Priority
Formal Def
Diagrams
Goal Diagram - Roadef 2005
To group for painting
To group to assemble
A request is ready for delivery when a order was received
Performance
Performance
Cars painted when a order was received
Cars assembled when they are painted in painting area
Operator
Cars transported when a order was received
Cars painted when they are on painting area
Cars grouped by special features
Transported for assembling when grouped
Cars assembled when they are grouped
Cars grouped according spraygun limit
Transported to painting when were grouped
The spray gun washed with solvent
The spray gun has painted when it was washed
Assembler
Painter
Performance
Performance
Performance
To wash spray gun
To Paint
To assemble
Transporter
Performance
Performance
To transport for painting
To transport for assembling

Fig. 10 Goal Model for the Car Sequencing Problem using RekPlan tool

cess called value co-creation) [4,31]. This new environment should treat a large amount of data to be applied to intelligent manufacturing design [47].

Therefore, the current demand in intelligent manufacturing design is to address product-service systems (PSS) [29] in the requirements cycle. In the perspective of Industry 4.0 the target artifact is a vast network of cyber-physical arrangements [4], which information source could also rely on big data [47]. However, the first step to studying and building proper tools to implement PSS's is formally exploring the requirements.

During the last decade, several software tools addressed the design of service-oriented systems: Enterprise Architect (Sparx) implemented SOMF, closely connected to information systems using a functional, object-oriented approach; a similar approach was used by Genesys (Vitech), using UML as semiformal and SysML as a formal presentation. However, these attempts did not cover the user's inclusion in the modeling and design, neither the final user nor the worker [48].

A goal-oriented approach was implemented by Objectiver (Respect-IT), introducing LTL as a formal presentation. It was used in this work as a general goal-oriented environment to feed the process-oriented approach. Thus, our proposal is based on using different practical systems connected by a transfer language as KML (for KAOS), PNML (for Petri Nets)[8]. The contribution is presenting the requirements development cycle based on a preliminary version of KML, presenting a practical tool to transfer requirements to Petri Nets: RekPlan. That could justify the goal-oriented as a feasible alternative to the functional approach in intelligent manufacturing.

PNML is used to transfer processes from RekPlan to Petri Net analysis tools as PIPE2, used in this work, to perform property analysis.

The transferrence algorithm from UML comes from a previous work of the same group [17]. It is inserted in the flexible cycle shown in Fig. 6. Scalability is still an issue, as it is for each tool we used, either commercial or academic. However, it applies to real problems, as shown in Fig. 11, even if not tested massively in manufacturing systems yet.

[8] PNML is part of the ISO/IEC Standard 15.909, that defines Petri Nets and its transfer language

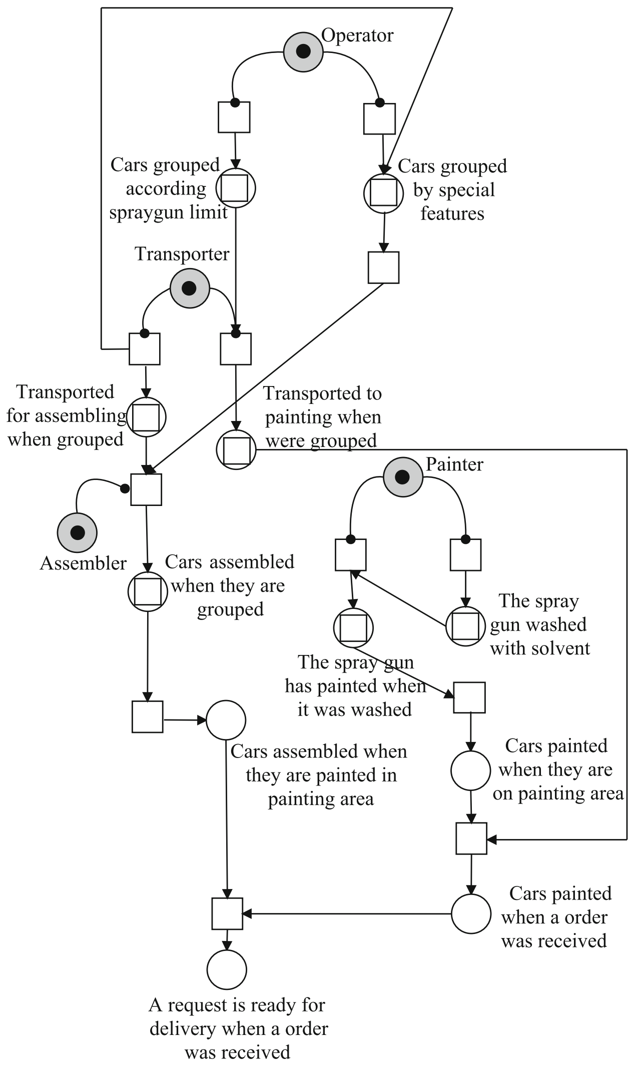

Operator
Cars grouped according spraygun limit
Cars grouped by special features
Transporter
Transported for assembling when grouped
Transported to painting when were grouped
Painter
Assembler
Cars assembled when they are grouped
The spray gun washed with solvent
The spray gun has painted when it was washed
Cars assembled when they are painted in painting area
Cars painted when they are on painting area
Cars painted when a order was received
A request is ready for delivery when a order was received

Fig. 11 Petri Net for Car Sequencing Problem

The critical point is that the proposal is prepared to face distributed production arrangements, can be evolved to service manufacturing, and include alternatives to the functional approach. Most importantly, it presents a clear formal development cycle for requirements, relying on invariants.

Another advantage is using Petri Nets as a formalism to intelligent manufacturing processes. Once Petri Nets typically represent processes in this area, it eliminates more transferrences from requirements to design.

6 Conclusion and further work

Systemic design of distributed intelligent manufacturing will demand formal approaches since the requirements phase, following a requirement life cycle, as depicted in Fig. 6. Different segments in the industry follow specific methods. Still, the growing complexity of manufacturing in technical organization and flexibility, besides the cloud's directions, will reinforce a tendency to treat requirements formally.

UML modeling is the primary approach used in legacy projects from the academy and industry. Therefore, any new alternative must comply with this legacy. However, the functional approach presents some problems for large systems, including distributed arrangements and service orientation, demanding the search for new alternatives.

Choosing a proper formal approach suitable to represent requirements and design of processes became a key issue. Petri Net can deal with dynamic processes and concurrency and was proposed to compose a model-based requirements development. Each enhancement is properly formalized and analyzed before evolving. It suits a systematic approach and is used to each component sub-system and the general arrangement.

Another critical point is that new challenges in intelligent manufacturing point to distributed arrangements of cyber-physical systems (CpF) or cloud-based CpFs [49]. Thus, it is crucial to search for alternatives to cover this demand. Finally, it is also essential to attend to the service-oriented flavor of new manufacturing systems.

The development cycle presented here covers only the basic and alternative methods to formalize and analyze requirements. However, a cognitive approach modeling user and value co-creation [50,51] for service manufacturing is being proposed as further work.

We are also worried about delivering a requirement engineering cycle that is usable and scalable to practical applications. Using different commercial and academic tools connected by a transferrence language based on XML we expect to reduce the time between new alternatives and their application.

Finally, we are developing a cloud-based environment to put together algorithms and tools already developed.

References

1. Kusiak, A.: Smart manufacturing. Int. J. Prod. Res. **56**(1–2), 508–517 (2017)
2. Kusiac, A.: Smart manufacturing must embrace big data. Nat. Comments. **544**(7648), 23–25 (2017)
3. Gershwin, S.B.: The future of manufacturing systems. Int. J. Prod. Res. **56**(1–2), 224–237 (2018)
4. Silva, J.R., Nof, S.Y.: Perspectives on manufacturing automation under the digital and cyber convergence. Polytechnica **1**(1–2), 36–47 (2018)
5. Koch, J., Michels, N., Reinhart, G.: Context model design for a process-oriented manufacturing change management. Procedia CIRP **41**, 33–38 (2016)
6. Silva, J.R., Nof, S.Y.: Manufacturing service: from e-work and service-oriented approach towards a product-service architecture. IFAC-PapersOnLine **48**(3), 1628–1633 (2015)

7. Beuren, F.H., Ferreira, M.G.G., Miguel, P.A.C.: Product-service systems: a literature review on integrated products and services. J. Clean. Prod. **47**(4), 222–231 (2013)
8. Isaksson, O., Larsson, T.B., Ronnback, A.O.: Development of product-service systems: challenges and opportunities for the manufacturing firm. J. Eng. Des. **20**(4), 329–348 (2009)
9. Pacaux-Lemoine, M.-P., Trentesaux, D., Rey, G.Z., Millot, P.: Designing intelligent manufacturing systems through human-machine cooperation principles: A human-centered approach. Comput. Ind. Eng. **111**, 581–595 (2017)
10. Song, W.: Customization-oriented Design of Product-service System. Springer, Singapore (2019)
11. Habrias, H., Frappier, M.: Software Specification Methods. Wiley, Hoboken, USA (2006)
12. Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications, vol. I. Wiley, UK (2009)
13. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using non-functional requirements. IEEE Trans. Softw. Eng. **18**(6), 483–497 (1992)
14. Silva, J.M., Silva, J.R.: A new hierarchical approach in requirements analysis of problems in automated planning. Eng. Appl. Artif. Intell. **81**, 373–386 (2019)
15. Cailliau, A.: Software requirements engineering: A risk-driven approach. PhD thesis, UCL-Université Catholique de Louvain, Belgium (2018)
16. Jureta, I., Faulkner, S., Thiran, F.: Dynamic requirements specification for adaptable and open service-oriented systems. In: Service-Oriented Computing - ICSOC 2007, pp. 270–282. Springer, Berlin, GE (2007)
17. Salmon, A.Z.O., del Foyo, P.M.G., Silva, J.R.: Verification of automated systems using invariants. In: Proceedings of the Brazilian Congress of Automation, pp. 3511–3518 (2014)
18. Silva, J.R., del Foyo, P.M.G.: Timed Petri Nets. Petri Nets: Manufacturing and Computer Science, pp. 359–372. Intech, (2012). Chap. 16
19. Linger, R.C., Mills, H.D., Witt, B.I.: Structured Programming. Theory and Practice. Addison-Wesley, Boston, USA (1979)
20. Girault, C., Valk, R.: Petri Nets for Systems Engineering. Springer, Berlin, GE (2003)
21. Baresi, L., Pezze, M.: On formalizing UML with high-level petri nets. In: Concurrent Object-oriented Programming and Petri Nets, pp. 276–304. Springer, Berlin, Heidelberg (2001)
22. Guerra, E., Lara, J.d.: A framework for the verification of uml models. examples using petri nets. JISBD 03: VIII Jornadas de Ingeniería del Software y Bases de Datos (2003)
23. Zhaoxia, H., Shatz, S.M.: Mapping uml diagrams to a petri net notation for system simulation. In: Proceedings of the 16th. Int. Conf. on Software Engineering & Knowledge Engineering, SEKE (2004)
24. Zhao, Y., Fan, Y., Bai, X., Wang, Y., Cai, H., Ding, W.: Towards formal verification of uml diagrams based on graph transformation. In: E-Commerce Technology for Dynamic E-Business, 2004. IEEE International Conference On, pp. 180–187 (2004). IEEE
25. Distefano, S., Scarpa, M.L., Puliafito, A.: From uml to petri nets: The pcm-based methodology. IEEE Trans. Softw. Eng. **37**(1), 65–79 (2011)
26. Wang, C.J., Fan, H.J., Pan, S.: Research on mapping uml to petri-nets in system modeling. MATEC Web. Conf. **44**, 1–4 (2016)
27. Gogolla, M., Hilken, F.: Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. Modellerung **2016**, 205–220 (2016)
28. Vasantha, G., Rajkumar, R., Lelah, L., Brissaud, D.: A review of product-service systems design methodologies. J. Eng. Des. **23**(9), 635–659 (2012)
29. Qu, M., Yu, S., Chen, D., Chu, J., Tian, B.: State-of-the-art of design, evaluation, and operation methodologies in product service systems. Comput. Ind. **77**, 1–14 (2016)
30. Qu, M., Yu, S., Chen, D., Chu, J., Tian, B.: Aircraft landing gear system: approaches with event-b to the modeling of an industrial system. Comput. Ind. **19**(2), 141–166 (2017)
31. Martinez, J., Silva, J.R.: Combining KAOS and GHENeSys in the requirement and analysis of service manufacturing. IFAC Proceedings Volumes (IFAC-PapersOnline) **48**(3), 1634–1639 (2015)
32. Rosa, N.R., Cunha, P.R., Justo, G.R.: An approach for resasoning and refining non-functional requirements. J. Braz. Comput. Soc. **10**(1), 62–84 (2004)
33. Himmler, F.: Function based requirements engineering and design - towards efficient and transparent plant engineering. In: Proceedings of the Int. Conf. on Current Trends in Theory and Practice of Informatics, SOFSEM 2015, pp. 436–448 (2015)
34. Lamsweerde, A.v.: Goal-oriented requirements engineering: a road trip from research to practice. In: Proceedings in 12th IEEE International. Requirements Engineering Conference (2004)
35. Mylopoulos, J., Chung, L., Yu, E.: From object-oriented to goal-oriented requirements. Commun. ACM. **31**(37), 31–37 (1998)
36. Hackel, R., Taentzer, G. (eds.): Graph Transformation, Specifications, and Nets. Lecture Notes in Computer Science, vol. 10800. Springer, Berlin, GE (2018)
37. Silva, M.: Half a century after carl adam petri's ph.d. thesis: A perspective on the field. Annual Reviews in Control 37, 191–219 (2013)
38. Yamaguchi, S.: Analysis of option to complete, proper completion and no dead tasks for acyclic free choice workflow nets. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E102–A**(2), 336–342 (2019)
39. Bernardinello, L., Lomazova, I.A., Nesterov, R., Pomello, L.: Soundness-preserving composition of synchronously and asynchronously interacting workflow net components. CoRR arXiv:2001.08064 (2020)
40. Hillah, L.M., Kordon, F., Petrucci, L., Tréves, N.: Pnml framework: An extendable reference implementation of the petri net markup language. In: Applications and Theory of Petri Nets, pp. 318–327. Springer, Bad Honef, GE (2010)
41. Dutra, D., Silva, J.R.: Product-service architecture (psa): toward a service engineering perspective in industry 4.0. IFAC-PapersOnLine **49**(31), 91–96 (2016)
42. Nguyen, A.: Challenge roadef 2005: Car sequencing problem. Online reference at http://challenge.roadef.org/2005/files/suite_industrielle_2005.pdf, last visited on August of 2016 **23** (2005). Brazilian Automation Society
43. Roy, R.: Sustainable product-service systems. Futures **32**(3–4), 289–299 (2000)
44. Niemann, M., Eckert, J., Repp, N., Steinmetz, R.: Towards a generic governance model for service oriented achitectures. In: Proceedings of Americas Conference on Information Systems, AMCIS 2008 (2008)
45. Bell, M.: Service-Oriented Modeling. John Wiley & Sons, Hobokebn, USA (2008)
46. Oliveira, V.C., Silva, J.R.: A service-oriented framework to the design of information system service. J. Serv. Sci. Res. **7**, 55–96 (2015)
47. Lee, J., Kao, H.-A., Yang, S.: Service innovation and smart analytics for industry 4.0 and big data environment. Procedia CIRP **16**, 3–8 (2014)
48. Lai, Z.-H., Tao, W., Leu, M.C., Yin, Z.: Smart augmented reality instructional system for mechanical assembly towards worker-centered intelligent manufacturing. J. Manuf. Syst. **55**, 69–81 (2020)

49. Silva, J.R., Vital, E.L.: Towards a formal design to service-oriented cloud manufacturing. In: Proceedings of the Brazilian Conf. on Automation (2020)
50. Pezzota, G., Cavalieri, S., Romero, D.: Engineering Value Co-cration in Product-Service Systems. IGI Global, Hershey, USA (2017)
51. Giesbrecht, T., Schwabe, G., Schenk, B.: Service encounters thinklets: how to empower service agents to put value co-creation into practice. Inf. Syst. J. **27**(2), 171–196 (2017)