IET Collaborative Intelligent Manufacturing

Research Article

IET Journals
The Institution of Engineering and Technology

Workflow composition and analysis in Industry 4.0 warehouse automation

eISSN 2516-8398
Received on 15th April 2019
Revised 8th August 2019
Accepted on 9th August 2019
E-First on 19th September 2019
doi: 10.1049/iet-cim.2019.0017
www.ietdl.org

Ajay Kattepur[1] ✉

[1]*Embedded Systems and Robotics, TCS Research and Innovation, Bangalore, India*
✉ *E-mail: ajay.kattepur@tcs.com*

Abstract: Workflow compositions have been exploited in business process modelling to handle concurrent invocations of modular components. With the emergence of Industry 4.0 warehouse automation, which enable the integration of business processes, mechanised robots, sensor–actuators and human participants, analysis and specification of workflows become crucial. As such environments have dynamic deployments due to varying demand rates and environmental conditions, the workflow compositions are intended to be adaptable to runtime changes. In addition, monitoring the end-to-end latency and optimal runtime binding is critical in industrial deployments such as warehouse automation. The authors provide specifications in the concurrent programming language Orc that supports most commonly used workflow patterns. Complex deployments involving multiple robotic agents and business processes further require analysis of correctness, liveness, and safety properties. In order to verify the workflows, the Orc specifications are translated into workflow net representations, with verification done using the TAPAAL model checker. The advantages of deploying fine grained analysis of workflows are demonstrated over picker/delivery robots involved in warehouse operations. The envisioned set of reusable specifications may be extended and applied to a variety of Industry 4.0 deployments to handle complex workflow interactions.

1 Introduction

Industry 4.0 [1] proposes the integration of robotics, cyber-physical systems, software services, and human participants with the following features:

i. *Interoperability*: Machines, internet of things (IoT) [2] devices, and humans connected and coordinating with each other.
ii. *Information transparency*: Physical systems enhanced with sensor data to create added value information systems.
iii. *Technical assistance*: This involves the use of intelligent devices to aid in informed decision making. Robotic automation may be identified to perform repetitive, unsafe or precise tasks.
iv. *Decentralised decisions*: The ability of such systems to make autonomous decisions; only critical cases will involve human intervention.

Warehouse automation [3] is an area of interest, as implementing the Industry 4.0 requirements would lead to significant monetary and performance benefits to large retailers. Current Industry 3.0 warehouse solutions are heavily dependent on centralised monitoring, human participants, and static deployments. Industry 4.0 warehouses are envisioned to move away from this with distributed/hybrid architectures, heavy automation, and adaptable deployments. In warehouses that may have hundreds of humans and robots on the shop floor [4], complex problem domains (scheduling, optimisation, and planning) require intricate handling of concurrent actions. High-level requirements will be translated to low-level task specifications, which would require formal workflow composition rules. Furthermore, concrete specifications of workflow descriptions are needed to hierarchically move from business requirements to executable component code for software/automated resources.

Workflow composition and analysis is an important area of work, receiving considerable scrutiny in the supply-chain, manufacturing, and business process communities. Such workflows are said to be grounded on a few basic patterns [5] based on which multiple standards and languages such as business process execution language (BPEL) [6] have been proposed. However, with the emergence of the internet of things (IoT) [2] and robotic process automation [7], such workflow specifications require enhancements to handle robotic and cyber-physical participants. Runtime binding of a set of services, seen in service-oriented architectures (SOAs) [8] must be integrated with other components such as robotic process automation software and autonomous robotic hardware.

Problems that need to be solved in the context of Industry 4.0 include workflow compositions such that individual agents (humans, robots, and servers) can satisfy a sub-part of the workflow process. In complex industrial workflows involving multiple modular components, the variation in invoking concurrent modules can affect the end-to-end quality of service (QoS) behaviour. This is in line with the Industry 4.0 Future Warehouse concept, where the centralised orchestrations provided by warehouse management systems (WMSs) [9] may be decentralised using autonomous interacting components [10]. In our work, we model the behaviours of cyber-physical systems, robotic processes/hardware, and human participants of workflows using multi-agent systems [11]. Integrating all such modular components into a unified specification framework is key to understanding trade-offs between invocation of services and resources needed in the form of robotic agents.

As a realistic use case, we employ multiple 'pickers-to-parts' autonomous robots in a warehouse setting, inspired by the Kiva robot delivery system [4] employed by Amazon (https://www.amazonrobotics.com/). In the 'pickers-to-parts' model, the conveyor belt-like robotic automation is replaced by a system of mobile autonomous robots that dynamically approach and pick parts in an industrial setup. We demonstrate that through the accurate use of workflow models, concurrent invocation and instantiation rates of such robots can be analysed. This can significantly impact the timing delays in stocking/procuring components from warehouses. Specifications are studied using the concurrent programming language Orc (https://orc.csres.utexas.edu/) [12], with high-level warehouse compositions interacting with individual picking/delivery robots. We extend this modelling with formal verification using translations to workflow net [13] models and the TAPAAL [14] model checking tool. Soundness, boundedness, and liveness

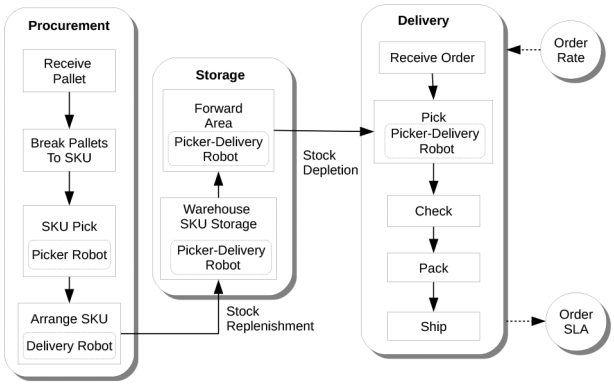

Fig. 1 *Tasks typically observed in warehouse distribution*

properties are studied for workflows that are safety critical. This would generate verifiable workflows that satisfy properties from the Industry 4.0 specifications [15]. The set of reusable workflow specifications would prove useful in a variety of automated industrial deployments. The fine-grained interactions further lead to latency monitors that may be tuned at runtime to provide marked improvements in completion times.

The principal contributions of this study are as follows:

i. Identify workflow models involved in Industry 4.0 warehouse procurement, storage, and delivery.
ii. Integrate modular robotic processes and computations to tackle warehouse workflow processes.
iii. Specify fine-grained (reusable) workflow compositions for warehouse automation involving concurrent services invocation, robotic specifications, and latency constraints in Orc.
iv. Formal verification through the use of model checkers that ensures soundness, liveness, and deadlock-freeness of Industry 4.0 workflows.
v. Demonstrate latency improvements provided by such fine grained workflow composition analysis on the end-to-end warehouse supply chain.

The rest of this paper is organised as follows: Section 2 provides an overview of processes involved in automated warehouse management, including robotic agents. The use and analysis of workflow patterns and eventual specifications in Orc are studied in Section 3. An overview of workflow verification, properties, and model checking is presented in Section 5. Section 4 specifies and analyses the automated warehouse workflow specifications with latency evaluation and verification. The evaluation of design-time and runtime executions of automated warehousing workflows, verification, and end-to-end latency analysis is studied in Section 6. This is followed by related work and conclusions in Sections 7 and 8.

2 Industry 4.0 warehouses

In this section, we present an overview of automated warehouses as well as autonomous robotic agents that are typically deployed in them.

2.1 Automated warehouses

Multi-product and multi-supplier warehouses play a critical role in most logistic supply chains [3]. Warehouses are used as a buffer to maintain excess product when there are variations in procurement on customer demand. They also allow consolidation of products in distribution centres located closer to delivery locations. Additionally, warehouses may perform ancillary activities such as breaking pellets into manageable units, labelling, and packaging of goods [3]. Typical multi-supplier warehouses have product receipt, storage, and delivery stages as described in Fig. 1:

i. *Procurement*: When a notification for goods arrival is processed, the goods must initially be checked for order integrity and damages. The products typically arrive in large pallets and may be broken down to prepare for storage. The broken down stock keeping units (SKUs) are then radio frequency identification (https://www.rfidjournal.com/logistics) tagged before being readied for storage.
ii. *Storage*: Picking and storage of SKUs must be handled efficiently. Of late, this is being automated with robotic pickers, forklifts, and delivery robots used in tandem [4]. The storage typically has a large storage area consisting of multiple racks. A subset of the stored products is moved to a forward area to ensure quicker delivery of products.
iii. *Delivery*: Once a delivery order is received, the products are procured from the warehouse. Typically, this process in handled by a centralised orchestrator that determines the number of pickers and time limits needed to procure products from the warehouse. Once all the required products are procured, they are collated and checked. Finally, packing and shipping of the product are done for order fulfilment.

As seen in Fig. 1, input order rates and the order service level agreements (SLAs) [16] are the parameters that may be monitored for such warehousing tasks. Multiple factors can affect this including the rate of supply of goods, forward area storage, and efficient storage locations. As picking and delivery tasks are being heavily automated by making use of robotic deployments [4], sequential or parallel invocation of these devices can have significant impact on the delay. Most warehouses intend to optimise these criteria to reduce flow time: the interval between order arrival and shipping.

Traditionally, the WMS [9] is tightly coupled with inventory management, order processing, and task assignment in warehouses. The WMS must be input with physical dimensions of all products, storage locations, and participants in the warehouse. It is a centralised system to orchestrate the flow of products, people, and machines on the warehouse floor. While such a system may work on a small scale, future warehouses involving increased automation and autonomy of participants need alternative models [4]. In multi-supplier, multi-party warehouses that involve multiple components, loose coupling of services would prove to be a superior architectural framework.

2.2 Robotic agents

In order to study the use of robotic automation in Industry 4.0 deployments at a high level of abstraction, we make use of intelligent agents. An intelligent agent perceives its environment through sensors, and determines the plan of actions to complete a task and acts upon the environment through effectors [17]. Robotic agents perceive the environment through camera/odometry sensors and make use of motor effectors to manipulate actions with respect to the environment. Typical agent actions, for instance with a part-picking robot, include

i. *Perception*: Camera sensors that sense pixels of varying intensity. This aids the robot in object detection and identification. Robot location, view, and environment may also be perceived.
ii. *Actions*: Pick up parts and sort into appropriate bins. Constraints may be placed on the robot capabilities and accuracy in performing such actions.
iii. *Goals*: Place parts in correct bins within the given time constraints.
iv. *Environment*: Warehouse environments with a conveyor belt or racks carrying parts.

Algorithm 1 (Fig. 2) presents an overview of an intelligent robot's perception and action via a knowledge base [17]. The knowledge base coordinates appropriate actions in relation to an individual robot's perception (for instance identifying the correct part). The knowledge base may also include descriptions of domain ontology, task templates, and resource descriptions. When a group of robots

1 **Input:** Robot Perception; Knowledge Base; Goal;
2 **Output:** Robot Action;
3 Knowledge Base ← *Update*(Knowledge-Base, Perception);
4 Action ← *Choose-Best-Action*(Knowledge-Base, Goal);
5 Knowledge Base ← *Update*(Knowledge-Base, Action);

Fig. 2 *Algorithm 1: intelligent robotic agent*

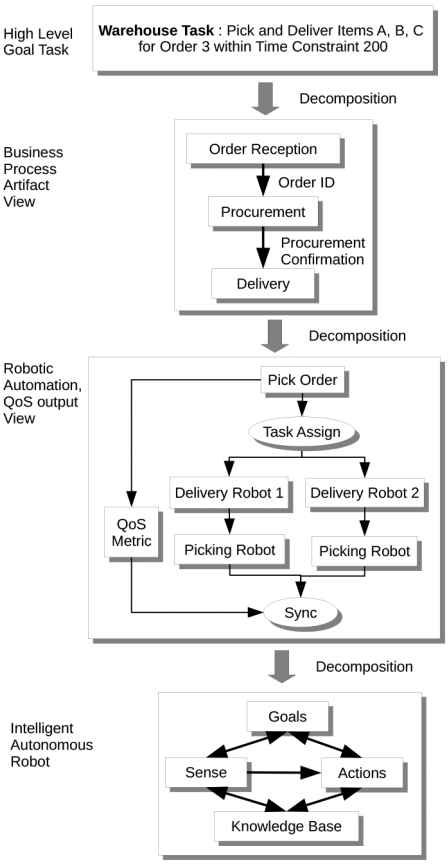

Fig. 3 *Decomposing high-level workflow tasks to business artefacts and robotic processes*

or a robot interacting with software/machines requires handling of complex tasks, an individual knowledge base would prove insufficient [4]. Additional workflow plans are needed to complete large tasks in a coordinated manner. This requires formal workflow specifications, described next.

Multi-agent systems [11] are typically employed to model interactions between heterogeneous, distributed and autonomous communicating devices. The autonomous agents have incomplete information and limited viewpoints of a global problem; consensus on the global task may be provided by sharing knowledge in a peer-to-peer or hierarchical fashion [11]. Data is also decentralised as agents may sense, store, and perform decision updates periodically. Such a decentralised, distributed architecture is crucial for large warehouses with dynamic delivery/order completion rates coupled with on demand resource provisioning and monitoring [10]. A practical implementation of such a system on the warehouse floor involves Amazon's Kiva robots [4] that autonomously move around a warehouse to aid in picking and delivery of goods.

When such multiple agents are involved within the picking and delivery workflow, it might be possible to allocate the entire task to a robot or create a multi-robot task. Typically, when multiple robotic agents are involved, the improvements are as follows: (i) *Speed*: A set of homogeneous delivery robots r_i, $i \in \{1, n\}$ would speed up the task by a factor of at most n. (ii) *Volume*: Each delivery/picking robot may have a limited number of items that can be conveyed at the required throughput. (iii) *Fault-tolerance*: Owing to redundancy in invoking multiple robots r_i, a single point of workflow failure is reduced considerably.

A requirement needed in multi-robot warehouse coordination is to have a unified specification framework to translate high-level requirements to low-level task specifications. This would involve studying concurrent workflow patterns and robot coordination that can be extensible to multiple deployment environments. In conjunction with concurrency, fine-grained analysis of latency constraints would also be factored into the model, which is essential for automated processes.

3 Workflow composition languages

Workflows integrating business processes, cyber-physical systems, and human participants have been extensively studied. This has received further gallop with the use of SOAs [8], in order to invoke modularly developed services. Industry standard workflow languages such as BPEL [6] have been produced. In this section, we look at techniques to hierarchically decompose high-level business process models to specifications that may be run on automated entities. Further formalisms on expressing workflows in Orc are also introduced.

3.1 Workflow decomposition

As specified in [5], workflows may be viewed in the following dimensions:

i. *Control flow*: Concerns the partial ordering of tasks executed in workflows.
ii. *Resource dimension*: Resources may be software, machines (robots, agents), and humans.
iii. *Case dimension*: Concerns the proper allocation of resources in order to satisfy constraints in a specific control-flow task.

Workflow specification languages allow the composition of modular systems in a formal way. This has been exploited to create distributed, verifiable deployments in SOA [8]. For our work, we primarily deal with the control and data flow perspectives of warehouse workflow execution. With complex tasks, task division may be done sequentially, in parallel, using the fastest response or in a hybrid manner.

The decomposition approach used is presented in Fig. 3, where a high-level task is subdivided and assigned to autonomous robotic/software resources. From a high-level organisational perspective, the tasks taken by the business processes follow the business artefact model [18], for instance, maintaining records of stock arriving or orders taken. In order to further decompose these high-level workflow processes, execution of robotic components using workflow constructs are provided. In all these cases, multiple constraints may be taken into account to decide which workflow to implement at runtime (price, latency, and resource utilisation). As we use intelligent agents [11] as the abstraction for robotic/machine components, tasks may be further allocated to such agents, who are able to sense, query knowledge bases and perform actions related to specific goals. Further elucidation of such tasks in the warehousing context is presented in the next section.

In automated deployments involving robotic/IoT components, additional constraints crop up in workflow specifications, including:

- *Hierarchical plans*: To deal with abstract business requirements, a high-level plan may be proposed that iteratively decomposes into executable instructions. For instance, a top-level plan may be ‘collect the item for delivery’, which is decomposed hierarchically until a picker robot has to execute `path plan`, `pick object`, and `deliver object` actions.
- *Complex control flow*: Invoking actions may be done using complex concurrent patterns such as synchronisation and data dependent trace executions. Verifying that the executed path confirms to specifications is an important aspect to consider in such cases.

- *Time*: Transactions and actions have time durations associated with them. Some may have hard constraints (robotic precision arm movement) while others may have soft constraints (delivery within 10 h with probability 0.8). Another difficulty is reasoning about compositions of such modular systems when integrated into larger workflows.
- *Resources*: Execution of workflows must consider limitations on the resources (human, robots) that may be employed and the total cost entailed in executions. Parallelising activities can have trade-offs that needed to be monitored and judiciously employed.

Algorithm 2 (Fig. 4) incorporates workflow execution plans within the context of intelligent robots (extension of Algorithm 1). Note that state based description of robotic agents is incorporated in order to provide efficient queries for goal completion. The first action in the workflow plan W is completed, with the next step iteratively allocated to the robotic agent driving it to iteratively complete the goal (lines 6–9 in Algorithm 2). If there are multiple robots allowed to execute a work item, the task allocation may be done via push (allocation of tasks) or pull (auctioning, parallel execution) control.

In order to formally specify such complex workflows, we make use of Orc [12], a concurrent programming language developed with wide area computations in mind.

3.2 Orc

In order to specify complex workflows, we make use of Orc [12], a concurrent programming language developed with wide area computations in mind. Orc has been shown to be able to specify complex workflow patterns [19], with the added advantage of being grounded in formal process-calculus and being able to deal with timing constraints. Orc may be translated into workflow nets [20] in order to verify liveness and safety properties that are crucial for industrial workflows.

The execution of workflows in Orc makes use of expressions, with the fundamental abstraction used in an Orc expression being a `site`. A site may be local or remote: for instance, a local site `add(x,y)` will provide the sum of two numbers; a remote site such as `GetStockValue(a)` would retrieve the stock value for a particular item. In order to create more complex workflow expressions based on `site` invocations, Orc makes use of the following combinators:

- *Parallel combinator* (|): Given two sites s_1 and s_2, the expression $s_1|s_2$ invokes both sites in parallel. The sites execute independently and the output published can be any of the outputs of s_1 or s_2.
- *Sequential combinator* ($\gg$): In the expression $s_1 \gg s_2$, site s_1 is evaluated initially. Every value published by s_1 initiates a separate execution of site s_2 with publications bound to any execution of s_2.
- *Pruning combinator* ($\ll$): In the expression $s_1 \ll s_2$, both sites s_1 and s_2 execute in parallel. If s_2 publishes a value, the execution of s_2 is terminated and the suspended parts of s_1 proceeds – this is the mechanism in Orc to block or terminate computations.
- *Otherwise combinator* (;): The expression $s_1; s_2$ first executes site s_1. If s_1 publishes no value and halts, then s_2 is executed instead. Halting is said to happen if all site calls with never publish any more values or will not call any more sites.

Two further expressions are used to aid in execution. The `signal` expression publishes a signal when executed and is similar to `if(true)`. The expression `stop` halts when executed as in `if(false)`. As Orc interacts with external sites, it has an inbuilt timer site `Rclock` to interact with the passage of time. This may be used to deal with timeouts when invoking remote sites.

As seen in Table 1, Orc sites and combinators may be applied to express all the patterns seen in workflows [5]. While simple patterns such as sequence and parallel split may be naively represented using the combinators, more complex patterns such as multi-merge and synchronisation require orchestration of sites, combinators, and timers.

Orc provides the ability to specify custom sites and functions, which proves suitable for specifying workflows in automated warehousing environments. It can also deal with timing constraints, concurrency, and scaling up/down of resources (using the object-oriented paradigm) that makes it useful in specifying Industry 4.0 workflows. Unlike other XML-based languages such as BPEL [6], Orc is based on a concurrent process calculus language that may be specified with object-oriented paradigms, allowing for modular, scalable, and reusable code. It also provides fine-grained timing constraints that are traditionally missing in other workflow specification languages. Furthermore, as Orc is based on a formal process calculus, it can be used to verify properties of industrial workflows. Orc specifications may be translated to Petri net formalisms that may be input to model checkers to analyse safety/liveness properties.

```
1  Input: Robot Perception; Knowledge-Base; State; Goal; Time t;
       Workflow;
2  Output: Robot Action;
3  Knowledge Base ← Update(Knowledge-Base, Agent(Perception, t));
4  State ← State-Description(Knowledge-Base, t);
5  G ← Query(Knowledge-Base, Goal-Query(t));
6  W ← Workflow(Knowledge-Base, G, State);
7  if W = Empty then
       Action ← Stop;
8  else
       Action ← First(W);
9      W ← Rest(W);
10 Knowledge Base ← Update(Knowledge-Base, Agent(Action,t));
11 t ← t+1;
```

Fig. 4 *Algorithm 2: intelligent robotic agent with a workflow plan*

Table 1 Workflow patterns [5] represented in Orc

Orc	Workflow pattern
$s_1 \gg s_2$	sequence, milestone
$s_1 \mid s_2$	parallel split, multiple instances
$v \ll (s_1 \mid s_2)$	exclusive choice, synchronising merge
$s_1; s_2$	implicit termination, cancel activity/case
(s_1, s_2)	synchronisation
$s_1 \mid s_2 \gg v$	simple merge, multi-choice, multi-merge
$s_1 \gg s_2 \gg s_1$	arbitrary cycles
`ift`	deferred choice, discriminator
`lock`	interleaved parallel routing

4 Warehouse workflow specifications

Current warehouses make use of a centralised WMS [9] to coordinate all activities. Industry 4.0 compliant warehouses involve robotic processes, hardware, and humans working in independent modes to fulfil tasks. In order to provide a framework for such interactions, we specify multiple sites using Orc based on the operations provided in Fig. 1. Note that we specify this using agent-oriented interactions [11] where agents may be software, robotic entities or humans. Humans and robots can interact with the workflow input/outputs and change behaviour accordingly. Coordination is done through the centralised workflow orchestration that received inputs/outputs from these deployed agents. This moves away from a central WMS architecture to a hierarchical/hybrid architecture incorporating autonomous components [10].

Fig. 5 presents a high-level view of our specification approach. While typical late-runtime binding of services has been well established in the SOA community [8], we append this with

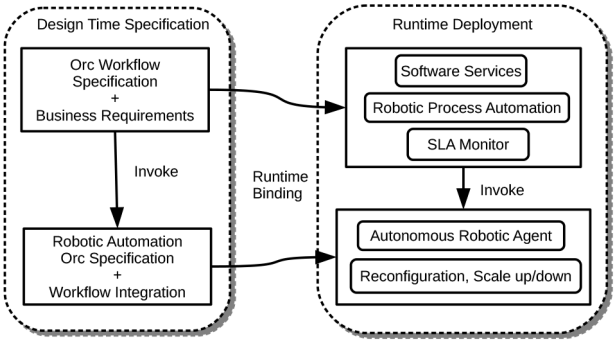

Fig. 5 *Design and runtime configurations in the warehouse workflow execution*

```
1  - Procurement site with pallet reception
2  def class Procurement()=
3      val PalletSize = Ref(5)
4      val SKU = Ref(500)
5      def ReceivePallet(ProdID) =  Dictionary() >rp>
           rp.prodID := ProdID
6      def BreakPallet(ProdID) = Dictionary() >bp>
           bp.palletsize := PalletSize? >> bp.SKUsize
           := SKU?
7      def SKUPick(ProdID, SKUSize) = Dictionary() >sp>
           sp.picksize := SKUSize
8      def ArrangeSKU(ProdID,SKUSize) = Dictionary()
           >ac> ac.picksize := SKUSize
9      stop
10
11 - Delivery site with order fulfillment
12 def class Delivery()=
13     def ReceiveOrder(OrderID) = Dictionary() >ro>
           ro.order := OrderID
14     def Pick(OrderID,size) = Dictionary() >po>
           po.size := size >> storage.RetrieveStock(
           OrderID,po.size?)
15     def Check(OrderID,size) = Dictionary() >co>
           co.size := size
16     def Pack(OrderID,size,checkstatus) = Dictionary()
            >po> po.check := checkstatus
17     def Ship(OrderID,size,checkstatus,packing) =
           Dictionary() >so> so.packed := packing
18     stop
19
20 - Storage site with inventory management
21 def class Storage()=
22     val PercentinForward = Ref(20)
23     val PercentinWarehouse = Ref(80)
24     val CurrentStock = Ref(100)
25     def WarehouseStorage(ProdID) = Dictionary() >wss>
            wss.percentage := PercentinWarehouse?
26     def ForwardStorage(ProdID) =  Dictionary() >fs>
           fs.percentage := PercentinForward?
27     def Stockvalue(ProdID,SKU) = SKU + CurrentStock?
           >p> CurrentStock := p
28     def RetrieveStock(ProdID,Units) = Dictionary()
           >rs> (Ift(Units <: 10) >> rs.stock := "
           Forward") | (Ift(Units :> 10) >> Rwait(d) >>
            rs.stock := "Warehouse")
29     stop
30
31 val Procurement = Procurement()
32 val Delivery = Delivery()
33 val Storage = Storage()
```

Fig. 6 *High-level warehouse activities implemented in Orc*

autonomous robotic components that are typically observed in Industry 4.0 industrial deployments. Note that both the warehouse workflow and the robotic components are specified in a unified framework in Orc at design time. This allows execution with various adaptation capabilities during runtime. The SOA-type services may be late-bound or replaced with other competing services from a registry to maintain SLAs. The autonomous robotic agents that are invoked in the workflow have some adaptation capabilities, with scaling-up/down of deployments possible depending on the varying demand rates. Specifying the interactions of all these components in a formal composition framework is necessary in order to

i. Reduce ambiguity in the execution of industrial workflows.
ii. Allow for fine-grained timing concurrency and timing analysis, specially necessary with robotic automation.
iii. Reuse specifications across multiple deployments with minimal changes in specifications.
iv. Ensure flexible deployments incorporating runtime adaptation and reconfiguration of modular components.

The architecture specified in Fig. 5 is hybrid in nature. While a centralised workflow orchestrator is used to coordinate the high-level warehousing tasks in a centralised manner, the presence of distributed autonomous entities such as robotic agents ensures adaptation and intelligent distribution of tasks. A unified framework to handle both high level and granular agent behaviour is necessary to maintain coherent execution in complex Industry 4.0 deployments.

We specify the functionalities of the procurement, storage, and delivery operations of a warehouse in Orc. In our specifications, we make use of the `def class` declaration that allows us to implement sites within Orc. The `def class` declaration provides encapsulation similar to classes in object-oriented programming. It also allows encapsulation of computations, methods, and resources relevant to sites managing those resources. Invocation makes use of the dot . access patterns for site calls. It consists of target expression E and key K. First, E is deflated to value v; if value v has a member named K, that member is published. Otherwise, the expression halts.

4.1 Warehouse workflow composition

The workflow compositions of the general description provided in Fig. 1 is given in Fig. 6. The general Orc abstractions used in the high-level specifications follow the declaration/instantiation expression (Fig. 7):

We explain some salient features of Fig. 6, which represents a concrete implementation of warehouse operations:

- *Procurement*: The re-usable site definition of the procurement workflow consists of functions to `ReceivePallet`, `BreakPallet`, `SKUPick`, and `ArrangeSKU`. This can be instantiated multiple times to receive products based on a `ProdID`, breakdown pallets from `PalletSize` to `SKU` and arrange them for storage. We make use of the `Dictionary()` site in Orc to create mutable maps from field names to values that may be extended. Latency incorporated into each of the sites may be specified using `Rwait` values. Assignment and retrieval from mutable references are done using `:=` and `?` patterns.
- *Delivery*: The site definition of the delivery workflow consists of functions to `ReceiveOrder`, `Pick`, `Check`, `Pack`, and `Ship`. Each step provides the input to the proceeding step, for instance, with parameters such as `checkstatus` and `packing`. Note that the definitions interact with the storage workflow using the `storage.RetrieveStock` site call to retrieve products.
- *Storage*: The storage site serves as the intermediary to the delivery and procurement workflows. It expresses the storage in the `WarehouseStorage` area and the `ForwardStorage` area. The current stock value is tracked in the `Stockvalue` site based on procurement and delivery. Note that another site `RetrieveStock` that procures smaller orders from the `ForwardStorage` area and larger orders (with additional delay) from the `WarehouseStorage`.

The advantage of encapsulating such sites in Orc is the ability to reuse such specifications across multiple warehousing deployments with minimal changes in composite specifications. Other workflow composition languages such as the XML-based BPEL are unable to efficiently support this code reuse in a new context.

```
-Warehouse Site Definition
def class WarehouseWorflow() =
   def Function() = (data, activity)
   stop

-Instantiate Runtime Warehouse Workflow Site
val WarehouseWorkflow = WarehouseWorflow()

-Invoke Warehouse Workflow
WarehouseWorkflow.Function instantiate
```

Fig. 7 *High Level Instance of a Warehouse Workflow in Orc*

```
-Robot Site Definition
def class Robot() =
   def RobotAgent() = (attributes, action)
   stop

-Instantiate Runtime Robot Site
val Robot = Robot()

-Robot Invocation in Workflow
Robot.RobotAgent instantiate
(WarehouseWorkflow activity, RobotAgent action)
```

Fig. 8 *High Level Instance of a Robot Workflow in Orc*

KUKA KR 40 PA Palletizing Robot
Max. Payload 40 kg, Max. Reach 1.8 m

KUKA KMP 1500 Mobile Platform
Max. Payload 1500 kg, Max. Velocity 1 m/s

Fig. 9 *KUKA warehouse automation picker and delivery robots*

```
-Picker Robot with payload, reach, delay
def class PickerRobot()=
   val Payload = 40
   val Reach = 180
   def Agent(ID) = Dictionary() >pr> pr.agentID := ID
         >> pr.agentModel := "KUKA KR 40 PA" >>
        Random(50) >v> Rwait(v) >> pr.agentdelay := v
         >> pr.payload = Payload >> pr.reach = Reach
   stop

-Delivery Robot with payload, velocity, delay
def class DeliveryRobot()=
   val Payload = Ref(0)
   val Velocity = 1
   val Distance = Random(100)
   def Agent(ID) = Dictionary() >dr> dr.agentID := ID
         >> dr.agentModel := "KUKA KMP 1500" >v>
        Rwait(v) >> dr.agentdelay := v >> dr.weight
        := 50 >> Payload := dr.weight?
   stop

val PickerRobot = PickerRobot()
val DeliveryRobot = DeliveryRobot()
```

Fig. 10 *Runtime robotic agent specification in Orc*

4.2 Robotic automation workflow composition

With automation being increasingly used for the delivery and picking tasks of the warehouse, robotic specification and invocation may be specified in Orc. We specify further reusable robotic sites, with the following attribute and instantiation expression (Fig. 8).

These may be physical robots such as those commercialised by Kuka (https://www.kuka.com/) and shown in Fig. 9. In Fig. 10, we define two sites called `PickerRobot` and `DeliveryRobot` that may be instantiated multiple times by the warehousing workflows using the `Agent(ID)` site call. Specifications for payload capacities, velocity, range, and delay characteristics are included. The

```
-Latency Increment Rules
def class LatencyQoS() =
  val counter = Ref(0)
  def min(t1, t2) = if  (t1 <: t2) then t1 else t2
  def max(t1, t2) = if  (t1 :> t2) then t1 else t2
  def sequential(t1, t2) = t1 + t2
  def join(t1, t2) = max(t1, t2)
  def best(t1, t2) = min(t1, t2)
  def increment(t1) = counter? + t1 >t> counter := t
       >> counter?
  stop

val Latency = LatencyQoS()
```

Fig. 11 *Latency specification in Orc*

advantage of such modelling is the hierarchical structure where the high-level warehouse workflow in Fig. 6 may be suitably deployed with autonomous robotic agents in Fig. 10 for run-time instantiation, binding, and resource adaptation (as shown in Fig. 2).

Note that though there are multiple activities such as path planning, optimal resource allocation, and robot-to-robot coordination that may be considered, we place emphasis primarily on the latency and concurrent task allocation among the robots. Payload, velocity, and dimensions of the robots may be specified using commercial data-sheets as in Fig. 9. As Orc is able to access external site Application Programming Interfaces (APIs) through JSON/RESTful calls, it may be integrated with robotic deployment environments such as ROS (http://www.ros.org/). It is also easy to extend this framework to represent agents such as humans and software that work autonomously but are coordinated by the workflow composition. Knowledge repositories such as RoboEarth [21] may be incorporated within this context to be applicable to other robotic deployments (describing robot capabilities, action partial orders, and environments).

4.3 Latency evaluation

As latency increments and timeouts are critical in industrial automation, integrating rules to update these constraints are necessary. QoS rules and algebraic expressions have been evaluated in Orc [22]. Such rules for latency increments are included, making use of the `LatencyQoS` site provided in Fig. 11. Note that the QoS increment rules are linked to the workflow patterns used and may be seamlessly integrated into the complex compositions. In Fig. 11, latency computations in fork-join operations may be done by making use of `Latency.join(t1,t2)` and that for selecting the fastest returning computation as `Latency.best(t1,t2)`. An operator called `Latency.increment(t1)` is provided to track latency increments during the computation of complex Orc expressions.

5 Workflow verification

In this section, we summarise properties that are typically used in verification of workflow nets. An overview of the TAPAAL model checker is also presented.

5.1 Workflow nets

In order to analyse and verify workflows integrated into automated environments (Algorithm 2), we translate Orc expressions into workflow nets, which have the advantage of being able to handle state-based descriptions of complex workflows [13]. Formally defining this

Definition 1: (net)**:** A Petri net is a triple (P, T, F)

- P is a finite set of places.
- T is a finite set of transitions ($P \cap T = \emptyset$).
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.

A place p is called an input place of a transition t iff there exists a directed arc from p to t. Place p is called an output place of transition t iff there exists a directed arc from t to p. The set of

input places to places and transitions are represented by $\bullet p$ and $\bullet t$. Similarly, $p\bullet$ and $t\bullet$ represent sets of places and transitions sharing common inputs. At any time a place contains zero or more tokens. The state or marking represents the distribution of tokens over places in the net: $M \in P \to \mathbb{N}$. A Petri net PN with initial state M is denoted as (PN, M). Transitions that fire given Petri net restrictions result in a new marking. Petri nets may be extended in the following ways to handle more complex modelling aspects:

- *Colour*: Coloured tokens may represent resources or attributes in the system. For instance, the flow of data such as order ID, vendor name, and object quality attributes may be incorporated with coloured tokens in Petri nets. Transitions describe the interactions between input and output tokens.
- *Time*: Most complex models require temporal behaviour such as durations and delays are to be incorporated. Time can be associated with tokens, places, and transitions.
- *Hierarchy*: A high-level Petri net may invoke a subnet hierarchically to decompose larger tasks to smaller subtasks. A subnet is an aggregate of a number of places, transitions, and subsystems.

5.2 Workflow net verification

A Petri net which models the control-flow dimension of a workflow is called a workflow net [13].

Definition 2: (Workflow net): A Petri net $\text{PN} = (P, T, F)$ is a Workflow net if:

i. There is a single source place $i \in P$ such that $\bullet i = \varnothing$.
ii. There is a single sink place $o \in P$ such that $o\bullet = \varnothing$.
iii. Every node $x \in P \times T$ is in the path from i to o.

A workflow net has one input place i and one output place o. While workflow nets may be used to directly model complex interactions, due to inherent limitations in data flow specifications, we use workflow nets as an intermediary representation. The following properties are typically employed to analyse workflow nets such that they correctly map requirement specifications to deployments [23]:

Definition 3: (Reachability): Given a workflow net $\text{PN} = (P, T, F)$ and a state M, a state M' is called reachable from M ($M \xrightarrow{*} M'$) iff there is a firing sequence σ such that $M \xrightarrow{\sigma} M'$.

Definition 4: (Liveness): A workflow net (PN, M) is live iff for every reachable state M' and every transition t, there exists a state M'' reachable from M' that enable t.

Definition 5: (Bounded): A workflow net (PN, M) is bounded iff for each place p, there is a natural number n such that, for every reachable state the number of tokens in p is less than n.

Definition 6: (Safe): A workflow net (PN, M) is safe iff for each place p, the maximum number of tokens does not exceed 1.

For the case of workflow nets, the additional condition is that the procedure will terminate eventually and the moment the procedure terminates, there is a token in place o and all the other places are empty. This is captured in the soundness property [23]:

Definition 7: (Soundness): A workflow net $\text{PN} = (P, T, F)$ is sound if and only if

i. For every state M reachable from state i, there exists a firing sequence leading from state M to state o.
ii. State o is the only state reachable from state i with at least one token in place o.
iii. There are no dead transitions in (PN, i).

We make use of model checkers to formally verify the above properties on Industry 4.0 workflows.

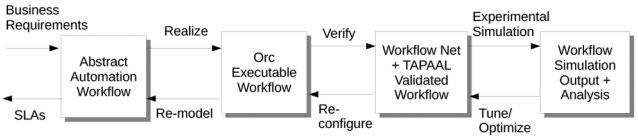

Fig. 12 *Workflow simulation, verification, and analysis*

5.3 Model checking workflows

The reason for formally specifying workflows is the advantage of verifying safety properties. This is critical, especially for complex industrial systems involving precision robotic/IoT components. Our model for integrating the hierarchy of Orc workflows into the verification is provided in Fig. 12. As Orc can be translated into workflow patterns, we perform model checking on workflow nets. Analysis of workflow nets is done using the TAPAAL model checker [14], with queries formulated in a subset of Computation Tree Logic (CTL):

i. `EF`: There exists some reachable marking that satisfies the property.
ii. `EG`: There exists a trace on which every marking satisfies the property.
iii. `AF`: On all traces there is eventually a marking that satisfies the property.
iv. `AG`: All reachable markings satisfy the property.

The predicate that needs to be checked is then specified using a combination of conjunction, disjunction, negation, and atomic propositions. The workflow analysis module in TAPAAL allows verification of soundness/liveness properties of workflow nets, including a detailed debugging information. Furthermore, search strategies such as depth first, breadth first or optimised search may be specified [14]. We now develop these techniques for deployment over automated warehouse workflows described in Orc.

5.4 Warehouse automation workflow verification

The rules for the translation of Orc expressions to workflow nets are based on the mapping in Table 1 [5, 19]. While simple patterns such as sequence and parallel split may be naively represented using the combinators, more complex patterns such as multi-merge and synchronisation require orchestration of sites, combinators, and timers. The use of workflow patterns ensures correct mapping between Orc specifications and the workflow net representation [19]. Further details on the verification of workflows may be found in [15].

Fig. 13*a* shows a high-level workflow net describing the automated warehouse order processing, procurement, and shipping transitions (sequence workflow pattern). Once the `Warehouse_Procure` transition is triggered, the picker and delivery automation workflows are enabled (synchronisation workflow pattern). The workflow net for the picker robot (Fig. 13*b*) includes `Task_Assignment`, robot `Pose_Estimate` and `Motion_Plan` (sequence and parallel split workflow pattern). The picker automation workflow completes by placing the object token in the delivery robot (synchronisation workflow pattern). Some of the expanded workflow steps are specific to robotic functioning, with details provided in [15]. This modelling shows how both business process workflows, robotic automation, and IoT sensors/actuators may be unified using the Orc and Workflow net modelling framework that we described in Section 3.

The workflow net model may be fed into model checkers for verification. In Industry 4.0 systems, verification is crucial to ensure correct execution of business automation requirements. Terms described in Section 5 with regard to soundness, liveness, and safety are crucial in automation workflows involving precision robotics interacting with business processes. The unified modelling also allows for composition of data flow and timing requirements in complex workflows, examined in the next section.

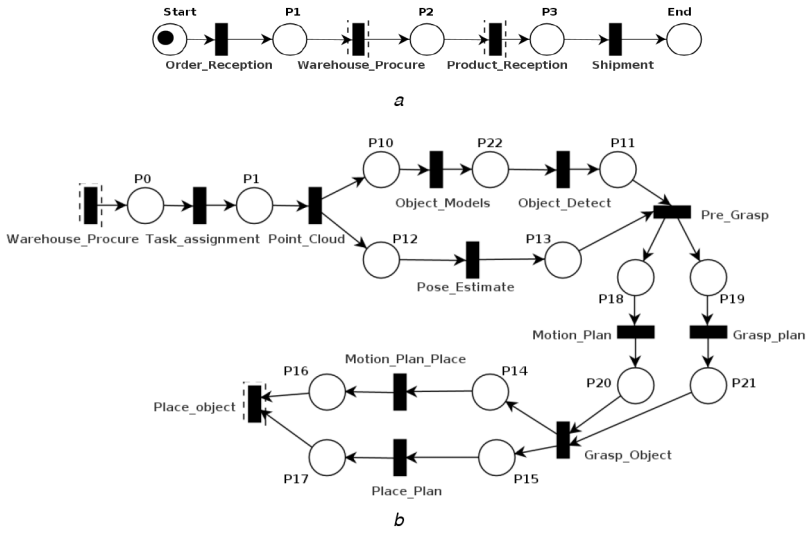

Fig. 13 *Workflow nets for*
(a) Warehouse workflow net (sequence pattern), ***(b)*** Picker automation workflow net (sequence, parallel split, synchronising merge patterns)

```
def ProcurementWorkflow() = procure.ReceivePallet(1) >> procure.BreakPallet(1)
>v>  (procure.SKUPick(1,v),PickerRobot.Agent("pr1")) >>
(procure.ArrangeCartons(1,v),DeliveryRobot.Agent("dr1")) >> storage.WarehouseSKUStorage(1)
>> storage.ForwardStorage(1) >> storage.Stockvalue(1,500) >> clock.time() >t>
Latency.increment(t) >v> Println("Procurement Time "+v)

def DeliveryWorkflow() = delivery.ReceiveOrder(1) >> DeliveryRobot.Agent("dr2") >>
(delivery.Pick(1,20),PickerRobot.Agent("pr2")) >> delivery.Check(1,5) >>
delivery.Pack(1,5,"ok") >>  delivery.Ship(1,5,"ok","Pack") >> storage.RetrieveStock(1,5) >>
clock.time() >t> Latency.increment(t) >v> Println("Delivery Time "+v)

ProcurementWorkflow() | DeliveryWorkflow()
```

```
Pallet Received ID: 1, Timestamp: 108
Order Received ID: 1, Timestamp: 113
Delivery Agent: dr2, Agent delay: 50, Timestamp: 114
Picking Agent: pr2, Agent delay: 35, Timestamp: 151
Picking Agent: pr1, Agent delay: 46, Timestamp: 156
Delivery Agent: dr1, Agent delay: 50, Timestamp: 611
Check Order ID: 1, Timestamp: 654
Pack Order ID: 1, Timestamp: 1154
Ship Order ID: 1, Timestamp: 1656
Stock from Forward Area for Order 1, Timestamp: 1656
Delivery Time 1657
signal
Procurement Time 3770
signal
```

Fig. 14 *Orc simulation output integrating warehouse workflows composed with automation timing analysis*

6 Simulation and analysis

In this section, we evaluate the use of workflow specification both for design time analysis as well as for runtime configurations and adaptation.

6.1 Orc execution of warehouse workflows

Design time execution and analysis of warehouse workflows is an important aspect of supply-chain planning. Typically, these are statically monitored/controlled through WMSs. Through our approach, a more flexible simulation-based environment may be generated, wherein modular workflows and robotic agents interact via specifications in Orc.

Fig. 14 shows the simulated output of an Orc programme that integrates workflow specifications and robotic agents. Two interleaved complex workflows are called in parallel: `ProcurementWorkflow() | DeliveryWorkflow()`. Each of these has site calls to multiple procurements, storage, and delivery activities specified in Fig. 6. Robotic piker and delivery agent instantiations are also incorporated to automate warehouse activities (Fig. 10). While each site may be implemented on distributed agents, our simulation implements the sites locally with exponentially distributed timing delays. Robotic delay parameters are taken from specifications in Fig. 4, with latency increment rules as specified in Fig. 11.

The output of a composite workflow simulation in Orc is shown in Fig. 14. We notice that various workflow activities proceed in the specified order. By the use of the `clock.time()` and `Latency.increment(t)` sites, the end-to-end procurement and delivery times may be tracked. This may be used to specify the order delivery SLAs as discussed in Fig. 1 (average/worst case bounds). Delays in procurement provided by the picking and delivery agents are also tracked within this framework. This framework allows the warehouse activity planner to study and analyse activities at design time. Specially in times with turbulent demand and supply rates, this may be combined with optimisation frameworks such as those proposed in [24] to efficiently manage inventory supplies.

6.2 Workflow verification

We next verify if the Orc workflow model confirms to specifications [15]. TAPAAL verification, when applied to the translated Workflow net model (Fig. 13), confirms that the workflow is `sound` and `bounded` by the number of tokens provided. Definitions provided in Section 5 on reachability, liveness, and safety are evaluated to be `true`. Using TAPAAL, the following specific CTL queries are further verified relating to the Industry 4.0 warehouse automation (Fig. 13 workflow net places/transitions referred):

i. *Workflow completion*: There exists some marking that satisfies state `Warehouse_Workflow.End` will be reached (Fig. 15).
ii. *Deadlock freeness*: On all traces, there is eventually a marking that satisfies absence of deadlocks (Fig. 16).
iii. *Proper termination*: All reachable markings satisfy the condition that reaching state `Warehouse_Workflow.End` implies all other states are empty (Fig. 17).
iv. *Violation absence*: On all traces, there is eventually a marking that satisfies an absence of tokens in multiple states (Fig. 18).

Such verification techniques guarantee verifiable correct deployments in Industry 4.0 warehouse automation.

6.3 Adaptive runtime executions

Another advantage of making use of workflow compositions is the intricate study of runtime executions. Consider a large picker/delivery task that requires coordination among multiple robotic agents (similar to Fig. 2). Typically, Amazon's Kiva system expects picking and delivery tasks to be completed within 20 min [4]. We consider the multiple models of invocation wherein two delivery robots 'dr1', 'dr2' approach a picker agent 'pr1' for a task (shown in Fig. 19).

We notice different instances of runtime execution within Orc: sequential, wherein the two delivery agents are assigned tasks one after the other; single, wherein the only one delivery agent is used for multiple tasks; parallel, wherein the delivery agents are invoked in parallel with a subset of tasks allocated; In the follower invocation pattern, if a task assigned to one of the delivery robots is not completed, an alternate robot is invoked; the fastest invocation pattern tasks are duplicated but accepted from the fastest serving robot.

Fig. 20 presents the simulated output of using a couple of agent invocation patterns in composition while incorporating the timeout constraint `Rwait`. In case the fastest pattern is unable to complete the delivery within the constraint, the parallel invocation pattern is invoked. It is due to such hierarchical, adaptable deployments of robotic agents that workflow compositions are of importance in complex deployments. Designers and planners of industrial warehouse workflows can integrate concurrency and timing behaviour in a unified framework through our specifications.

In order to provide a quantitative comparison of the schemes, we consider delivery agent robots with exponential completion times having mean of 5 min. The cumulative density of distributions collected after 10,000 Monte–Carlo runs (using `clock.time()` in Orc) are shown in Fig. 21. As expected, the parallel and fastest workflow specifications outrun the single workflow execution. While the follower execution initially overlaps the single execution times, once the threshold of 5 min is exceeded, it improves to the sequential execution times. The percentile values have also been displayed with the 95th percentiles provided with the worst being single robot tasks allocation having 30 min latency and the best being fastest 14.2 min (52.6% improvement), parallel 18.8 min (37.7% improvement) and sequential/follower 24 min (20% improvement). With Amazon's Kiva robots aiming for 15–20 min turnaround times [4], such accurate analysis and workflow adaptation are crucial to ensure SLAs. Note that while we recommend Monte–Carlo runs be performed a-priori with statistical inputs, this may be replaced with runtime monitoring and reconfiguration triggers based on thresholds.

6.4 Warehouse automation timing analysis

Workflows in Orc allow for intricate timing analysis of end-to-end systems. This may also be integrated as hard timeouts that must be adhered to during workflow execution. We examine the timing guards to be provided to the end-to-end warehouse, time to pick an assigned object by the picker robot and deadline for the delivery robot to reach the picking destination. Setting these deadlines involves simulation analysis of Orc workflows (Fig. 6).

Fig. 22 shows the composed latency bounds produced for various workflows in the automated warehouse example. Mean

```
1  - Workflow Completion
2  EF Warehouse_Workflow.End = 1
```

Fig. 15 *Verifying Workflow Completion in TAPAAL*

```
1  - Deadlock Freeness
2  AF !(deadlock)
```

Fig. 16 *Verifying Deadlock Freeness in TAPAAL*

```
1  - Proper Termination
2  AG (Warehouse_Workflow.End = 0 or (
       Warehouse_Workflow.Start = 0 and
       Picker_Robot.P0...P22 = 0 and
       Delivery_Robot.P3...P22 = 0))
```

Fig. 17 *Verifying Proper Termination in TAPAAL*

```
1  - Violation Absence
2  AF !(Warehouse_Workflow.P1 >= 1 and (Picker_Robot.P0
       >= 1 or Delivery_Robot.P3 >= 1))
```

Fig. 18 *Verifying Absence of Violation in TAPAAL*

```
1  - Sequential Invocation of Agents
2  def Sequential() = PickerRobot.Agent("pr1") >>
       DeliveryRobot.Agent("dr1") >>
       DeliveryRobot.Agent("dr2")
3
4  - Single Agent Invocation
5  def Single() = PickerRobot.Agent("pr1") >>
       DeliveryRobot.Agent("dr1") >>
       DeliveryRobot.Agent("dr1")
6
7  - Parallel Invocation of Agents (Task Division)
8  def Parallel() = PickerRobot.Agent("pr1") >>
9      (DeliveryRobot.Agent("dr1"), DeliveryRobot.Agent(
           "dr2"))
10
11 - Agent Invocation with Follower Agent
12 def Follower() = PickerRobot.Agent("pr1") >>
13     (DeliveryRobot.Agent("dr1") | Rwait(t)) >> stop ;
           DeliveryRobot.Agent("dr2")
14
15 - Pruning Invocation of Agents (Fastest Selected)
16 def Fastest() = PickerRobot.Agent("pr1") >> (v << (
       DeliveryRobot.Agent("dr1") | DeliveryRobot.Agent
       ("dr2")))
```

Fig. 19 *Orc expressions with robotic workflows*

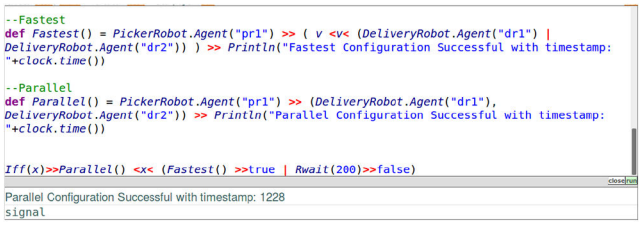

Fig. 20 *Composite workflows involving timeouts and concurrent invocation of agents*

values of transitions (exponentially distributed) are considered with 10,000 Monte–Carlo runs of the Orc simulator. We notice that for both `Picker` and `Delivery`, mean latencies are similar. Improvement in mean latency of each workflow site/transition (Figs. 6 and 13) from 10 to 5 min improves the latency deadline of the 95th percentile by ~50%. This is reflected in the end-to-end warehouse workflow that also improves the overall distribution. The advantage of such a model is that the hard timing constraints/improvements provided by the robots may be composed as soft contractual guarantees in the SLA. For instance, if the Picker and Placer robots complete tasks within 80 min, the order will be

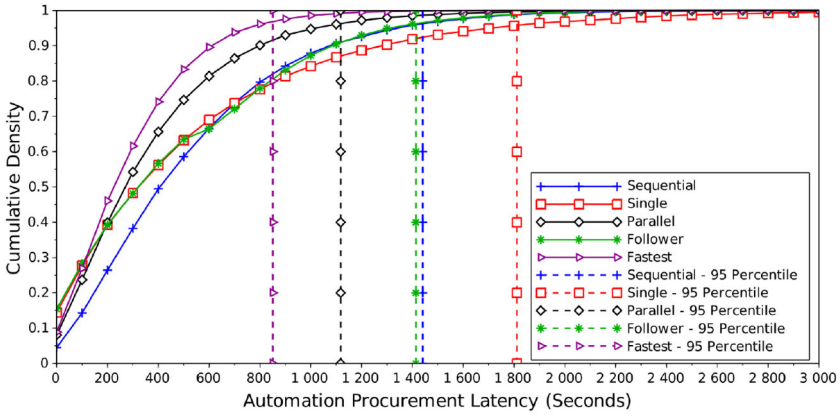

Fig. 21 *Delay distributions with varying runtime workflow compositions*

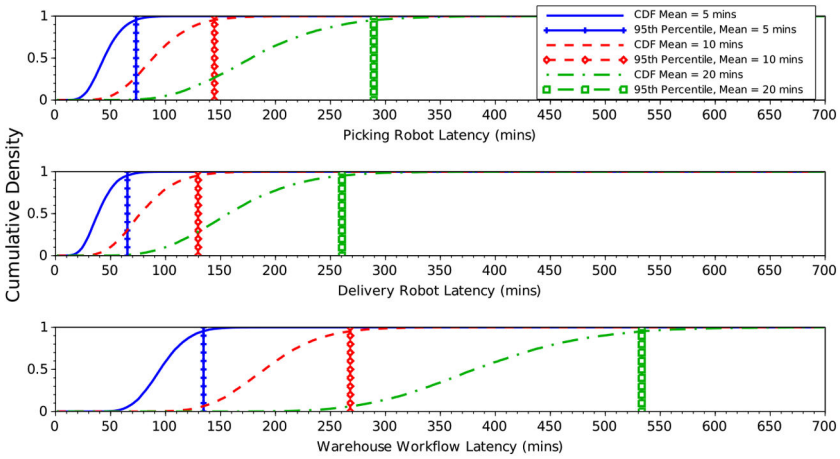

Fig. 22 *Composed latency distributions for picker, delivery and warehouse operation workflows*

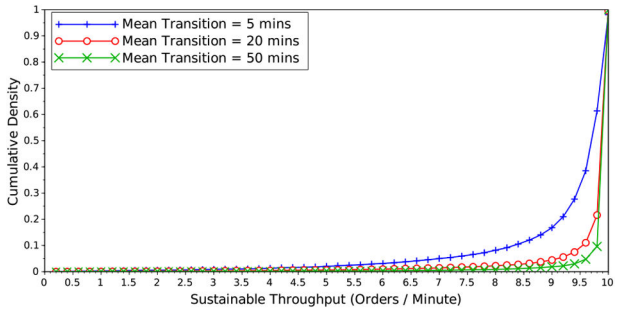

Fig. 23 *Throughput supported by the warehouse*

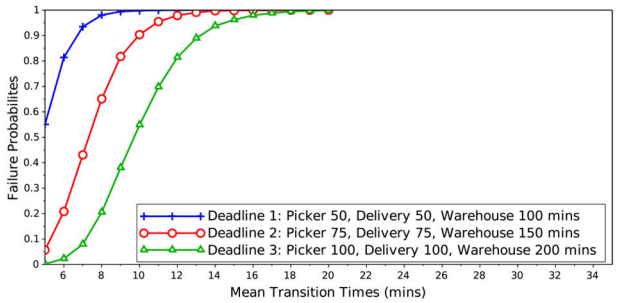

Fig. 24 *Failure probabilities versus mean transition times and deadlines*

successfully delivered to the customers within 100 min in 70% and within 135 min in 95% of cases.

Fig. 23 shows the trade-off throughput that may be supported by the warehouse when incoming orders/minute are increased. We make use of the utilisation law [25] to derive the peak throughput supported

$$\text{Peak throughput} = \frac{\text{Peak utilisation}}{\text{Operating latency bound}} \quad (1)$$

As expected, for shorter delivery times, the number of order/minutes is limited. For instance, with a rate of 9 orders/minute it would be prudent to move to transitions with 20 min deadlines to ensure SLAs are met. Fig. 24 shows the failure probabilities for various mean transition times (exponential distributions) and deadlines, which is another view of the workflow execution. We observe lower failure probabilities with extended deadlines coupled with smaller mean transition times. For instance, for mean transition times of 8 min, the failure probability of deadline 3 (picker 100 min, placer 100 min, warehouse 200 min) is 0.2, which is a vast improvement over the 0.65 for deadline 2 (picker 75 min, placer 75 min, warehouse 150 min) and 0.97 for deadline 1 (picker 50 min, placer 50 min, warehouse 100 min).

This demonstrates the end-to-end modelling, simulation, verification, and latency analysis of Industry 4.0 warehouse workflows.

7 Related work

In this section, we provide an overview of related work in Industry 4.0 warehouses, robotic automation and workflow modelling.

7.1 Industry 4.0 warehouses

Industry 4.0 [1] requires increased automation, autonomy and adaptation among distributed entities working in factory/warehousing environments. A central entity for control and coordination in warehouses has traditionally been the WMS [3, 9]. However, with increased warehouse automation such as those demonstrated with Amazon's Kiva robots [4], distributed and decentralised monitoring and control of these components are needed.

While traditional warehouses only require orchestration of business processes, automated warehouses include intelligent robotic agents and IoT devices [17], requiring accurate workflow models. In industrial environments where software and (mobile) hardware components have tight interactions, such workflow specifications would involve intricate flow control and concurrency issues. In [26], the warehouse automation throughput is increased by up to 10% by optimising inventory forward area stock and maximising pick rate velocities. Petri net model of factory workflows is presented in [27], using which properties such as liveness and deadlock freeness are synthesised. A simulation framework is further proposed in [28], where design of the warehouses and deployment of automated guided vehicles may be coordinated. In [26], the warehouse automation throughput is optimised by optimising inventory forward area stock and maximising pick rate velocities – shown to increase throughput by up to 10% over the baseline.

7.2 Workflow modelling and analysis

The use of formal compositions and concurrent programming has been well exploited in the SOA community [6, 8]. With robotic process automation [7] increasingly coming into the forefront, linking robotic processes and hardware within this framework opens up the possibility to analyse heterogeneous interactions. An instance of such work is [24], where the Dell logistic processes are updated with optimisation services to dynamically vary resupply rates.

The interactions may be specified using workflow semantics which follows certain patterns [5]. The integration of distributed computing paradigms and workflow compositions has further come into the forefront with development of programming languages such as Orc [12, 19]. Orc is able to handle all the workflow patterns [5] seen in complex workflow specifications. Orc may be translated into workflow nets [20] in order to verify liveness and safety properties that are crucial for industrial workflows. In industrial environments where software and (mobile) hardware components have tight interactions, such workflow specifications would involve intricate flow control and concurrency issues. Algebraic rules for QoS increments have been specified in [22], which is integrated into our model.

7.3 Robotic agent workflows

Multi-agent systems [11] provide the right level of abstraction to study and reason about autonomous components in complex deployments. In [29], concurrent patterns such as periodic timers and active objects are integrated into robotic task executions. Cloud robotics [30] is another improvement in traditional multi-robot deployments, with cloud-based knowledge repositories such as RoboEarth [21].

The use of workflow models in industrial settings involving IoT, robotics, and software systems have started receiving some attention. In [31], a model-based programming environment is proposed that integrates robotic control flow with workflows specified as structured flow charts. A visual workflow simulation environment is proposed in [32] that helps integrate robotic and human participants in Industry 4.0 workflows. In [29], concurrent patterns such as periodic timers and active objects are integrated into robotic task executions. The use of workflows to integrate IoT sense–compute–actuate devices is proposed in [33]. In [34], workflow patterns are analysed using queuing network models to generate end-to-end performance guarantees.

In this study, we integrate reusable workflow specifications with warehouse automation to ensure efficient and adaptable runtime deployments. This would form a basis for design time specifications across multiple Industry 4.0 warehouse scenarios. The model has been studied in terms of modularity, reuse, latency improvements, and verification of safety/liveness properties.

8 Conclusions

Industry 4.0 automated warehouses involve a myriad of heterogeneous participants including WMSs, picking/delivery robots, and human agents. Integrating them into a unified framework will involve analysis of multiple concurrent processes, requiring workflow modelling and optimisation. In this work, we integrate many such autonomous components by specifying composite workflows in Orc. We demonstrate that complex interactions and workflow patterns may be suitably instantiated within this framework. The workflows are translated into workflow net models to allow model checkers to analyse safety, soundness, and liveness properties. Accurate timing analysis and bounds are then studied, which allows bottlenecks and throughput improvements to be identified in complex workflows. Analysis of latency constraints in case of automated warehouse/picking delivery shows that runtime improvements may be observed through adaptive switching between workflow invocation patterns. Such a set of reusable libraries would prove invaluable across industrial warehouse management activities both for design time configuration and runtime analysis.

In the future, we would like to integrate more low-level details of automated warehouses including robot location, resource utilisation, and delivery tracking.

9 References

[1] Hermann, M., Pentek, T., Otto, B.: 'Design principles for industrie 4.0 scenarios'. 49th Hawaii Int. Conf. on System Sciences, Washington, DC, USA, 2016

[2] Greengard, S.: '*The internet of things*' (MIT, Cambridge, MA, USA, 2015)

[3] Bartholdi, J., Hackman, S.: '*Warehouse and distribution science*' (The Supply Chain and Logistics Institute, Georgia Institute of Technology, Georgia, 2016)

[4] Wurman, P., D'Andrea, R., Mountz, M.: 'Coordinating hundreds of cooperative, autonomous vehicles in warehouses', *AAAI Artif. Intell. Mag.*, 2008, **29**, (1), pp. 9–19

[5] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: '*Workflow patterns: the definitive guide*' (MIT Press, Cambridge, MA, USA, 2016)

[6] Pant, K., Juric, M.B.: '*Business process driven SOA using BPMN and BPEL*' (Packt Publishing Ltd, Birmingham, UK, 2008)

[7] van der Aalst, W., Bichler, M., Heinzl, A.: 'Robotic process automation', *Bus. Inf. Syst. Eng.*, 2018, **60**, (4), pp. 269–272

[8] Erl, T.: '*Service-oriented architecture: concepts, technology, and design*' (Prentice Hall, Upper Saddle River, NJ, USA, 2005)

[9] JDA Warehouse Management System (WMS): Available at https://jda.com/solutions/profitable-omni-channel-retail-solutions/intelligent-fulfillment/warehouse-management, 2018

[10] Kim, B., Graves, R., Heragu, S., *et al.*: 'Intelligent agent modeling of an industrial warehousing problem', *IIE Trans.*, 2002, **34**, pp. 601–612

[11] Shoham, Y., Leyton-Brown, K.: '*Multiagent systems: algorithmic, game-theoretic, and logical foundations*' (Cambridge University Press, New York, NY, USA, 2009)

[12] Kitchin, D., Quark, A., Cook, W., *et al.*: 'The Orc programming language'. Proc. FMOODS/FORTE, Lisboa, Portugal, 2009 (LNCS, **5522**), pp. 1–25

[13] van der Aalst, W.M.P.: 'Three good reasons for using a Petri-net-based workflow management system', in Wakayama, T., Kannapan, S., Khoong, C.M., *et al.* (Eds): '*Springer information and process integration in enterprises*' (Springer, Boston, MA, USA, 1998), pp. 161–182

[14] Byg, J., Joergensen, K.Y., Srba, J.: 'TAPAAL: editor, simulator and verifier of timed-arc Petri nets'. 7th Int. Symp. on Automated Technology for Verification and Analysis, Macao, China, 2009 (LNCS, **5799**), pp. 84–89

[15] Kattepur, A., Mukherjee, A., Balamuralidhar, P.: 'Verification and timing analysis of Industry 4.0 warehouse automation workflows'. IEEE 23rd Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 2018

[16] Jin, L., Machiraju, V., Sahai, A.: 'Analysis on service level agreement of web services', HP Laboratories, Palo Alto, 2002

[17] Russell, S., Norvig, P.: '*Artificial intelligence: a modern approach*' (Pearson, Essex, UK, 2009, 3rd edn.)

[18] Nigam, A., Caswell, N.S.: 'Business artifacts: an approach to operational specification', *IBM Syst. J.*, 2003, **42**, (3), pp. 428–445

[19] Cook, W.R., Patwardhan, S., Misra, J.: 'Workflow patterns in Orc'. Proc. Int. Conf. on Coordination Models and Languages (COORDINATION), Bologna, Italy, 2006

[20] Kotb, Y., Beauchemin, S., Barron, J.: 'Workflow nets for multiagent cooperation', *IEEE Trans. Autom. Sci. Eng.*, 2012, **9**, (1), pp. 198–203

[21] Tenorth, M., Perzylo, A., Lafrenz, R., *et al.*: 'Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth framework', *IEEE Trans. Autom. Sci. Eng.*, 2013, **10**, (3), pp. 643–651

[22] Benveniste, A., Jard, C., Kattepur, A., *et al.*: 'QoS-aware management of monotonic service orchestrations', *Form. Methods Syst. Des.*, 2013, **44**, (1), pp. 1–43

[23] van der Aalst, W.M.P.: 'Workflow verification: finding control-flow errors using Petri-net-based techniques'. Business Process Management, Berlin, Heidelberg, Germany, 2000 (LNCS, **1806**), pp. 161–183

[24] Kattepur, A., Benveniste, A., Jard, C.: 'Optimizing decisions in web services orchestrations'. Int. Conf. on Service Oriented Computing (ICSOC), Paphos, Cyprus, 2011, pp. 77–91

[25] Lazowska, E.D., Zahorjan, J., Graham, G.S., *et al.*: '*Quantitative system performance*' (Prentice Hall, Upper Saddle River, NJ, USA, 1984)

[26] Stowe, J.D.: 'Throughput optimization of multi-agent robotic automated warehouses', Massachusetts Institute of Technology, 2016
[27] Basile, F., Chiacchio, P., Coppola, J.: 'A hybrid model of complex automated warehouse systems – part I: modeling and simulation', *IEEE Trans. Autom. Sci. Eng.*, 2012, **9**, (4), pp. 640–653
[28] Cossentino, M., Lodato, C., Lopes, S.*, et al.*: 'Multi agent simulation for decision making in warehouse management'. Federated Conf. on Computer Science and Information Systems (FedCSIS), Szczecin, Poland, 2011
[29] Rusakov, A., Shin, J., Meyer, B.: 'Concurrency patterns for easier robotic coordination'. Int. Conf. on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015
[30] Hu, G., Tay, W., Wen, Y.: 'Cloud robotics: architecture, challenges and applications', *IEEE Netw.*, 2012, **26**, (3), pp. 21–28
[31] Geisinger, M., Barner, S., Wojtczyk, M.*, et al.*: 'A software architecture for model-based programming of robot systems', in Kröger, T., Wahl, F.M. (Eds): '*Springer advances in robotics research*' (Springer, Berlin, Heidelberg, Germany, 2009), pp. 135–146
[32] Dukalski, R., Cencen, A., Aschenbrenner, D.*, et al.*: 'Portable rapid visual workflow simulation tool for human robot coproduction'. 27th Int. Conf. on Flexible Automation and Intelligent Manufacturing, Italy, 2017
[33] Seiger, R., Assmann, U., Huber, S.: 'A case study for workflow-based automation in the internet of things'. IEEE Int. Conf. on Software Architecture Companion (ICSA-C), Seattle, WA, USA, 2018
[34] Kattepur, A.: 'Towards structured performance analysis of industry 4.0 workflow automation resources'. ACM/SPEC Int. Conf. on Performance Engineering, Mumbai, India, 2019